VisualAge Generator

Design Guide

Version 4.0

SH23-0264-00

Note

Before using this document, read the general information under “Notices” on page ix.

Second Edition (October 2000)

This edition applies to the following licensed programs:
- IBM VisualAge Generator Developer for OS/2 and Windows NT Version 4.5
- IBM VisualAge Generator Server for OS/2, AIX, Windows NT, HP-UX, and Solaris Version 4.5
- IBM VisualAge Generator Server for AS/400 Version 3.1
- IBM VisualAge Generator Server for AS/400 Version 3.6
- IBM VisualAge Generator Server for MVS, VSE, and VM Version 1.2

Order publications by phone or fax. IBM Software Manufacturing Solutions takes publication orders between 8:30 a.m. and 7:00 p.m. eastern standard time (EST). The phone number is (800) 879-2755. The fax number is (800) 445-9269. Faxes should be sent Attn: Publications, 3rd floor.

You can also order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. You can send your comments in any one of the following methods:

Electronically, using the online reader comment form at the address listed below. Be sure to include your entire network address if you wish a reply.
- http://www.ibm.com/software/ad/visgen

By mail to the following address:

IBM Corporation, Attn: Information Development, Department G7IA Building 062, P.O. Box 12195, Research Triangle Park, NC 27709-2195.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood NY 10594, U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact the SWS General Legal Counsel, IBM Corporation, Department TL3 Building 062, P. O. Box 12195, Research Triangle Park, NC 27709-2195. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

IBM has made reasonable efforts to ensure the accuracy of the information contained in this publication. If a softcopy of this publication is provided to you with the product, you should consider the information contained in the softcopy version the most recent and most accurate. 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.

IBM may change this publication, the product described herein, or both.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

ACF/VTAM
AD/Cycle
AIX
AS/400
CICS
CICS/ESA
CICS/MVS
CICS OS/2
CICS/VSE
CICS/6000
CICS for Solaris
COBOL/370
COBOL/400
Common User Access
CUA
DATABASE 2
DataJoiner
DB2
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
Operating System/2
IBM
IMS
IMS/ESA
Language Environment
MVS
OS/2
OS/400
SAA
SQL/DS
SQL/400
System/370
Virtual Machine/Enterprise Systems Architecture
VisualGen
VisualAge
VM/ESA
VSE/ESA

The following terms are trademarks of other companies:

Adobe	Adobe Systems, Inc
Apollo	Apollo Computer, Inc
C++	American Telephone & Telegraph Company
COBOL Workbench	Micro Focus Limited
DDS	Sony Corporation
DIF	Lotus Development Corporation
EDA/SQL	Information Builders, Inc.
Express	Parasoft Corporation
HP-UX	Hewlett-Packard Company
Legend	Sigma Designs Inc.
Micro Focus	Micro Focus Limited
Micro Focus IMS Option	Micro Focus Limited
OSF/Motif	Open Software Foundation, Inc.
Oracle	Oracle Corporation.
Solaris	Sun Microsystems, Inc.
Sybase	Sybase Inc.
UNIX	X/Open Company Ltd.
Windows NT	Microsoft Corporation
X/Open	X/Open Company Ltd.

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Terminology used in this document

Unless otherwise noted in this publication, the following references apply:

- MVS CICS applies to Customer Information Control System/Enterprise Systems Architecture (CICS/ESA) systems.

- CICS applies to CICS for VSE/ESA, CICS/ESA, CICS for OS/2, CICS for AIX, CICS for Windows NT, and CICS for Solaris.
- CICS for Windows NT refers to IBM TXSeries for Windows NT Version 4.2.
- CICS for AIX refers to IBM TXSeries for AIX Version 4.2.
- CICS for Solaris refers to IBM WebSphere Enterprise Edition Version 3.0.
- IMS/VS applies to Information Management System/Enterprise System Architecture (IMS/ESA) and IMS/ESA Transaction Manager systems.
- IMS applies to IMS/ESA and IMS/ESA Transaction Manager, and to message processing program (MPP), IMS Fast Path (IFP), and batch message processing (BMP) regions. IMS/VS is used to distinguish MPP and IFP regions from the IMS BMP target environment.
- LE applies to the IBM Language Environment for MVS and VM.
- COBOL applies to any of the following types of COBOL:
  - IBM VisualAge for COBOL for OS/2
  - ILE COBOL/400
  - IBM COBOL for VSE
  - IBM COBOL for MVS and VM
- "Region" and "CICS region" correspond to the following:
  - CICS for MVS/ESA region
  - IMS region
  - CICS for VSE/ESA partition
  - CICS for OS/2 system
  - CICS for AIX system
  - CICS for Windows NT system
  - CICS for Solaris system
- DB2/VSE refers to SQL/DS Version 3 Release 4 or later. Any references to SQL/DS refer to DB2/VSE and SQL/DS on VM. In addition, any references to SQL/400 refer to DB2/400.
- OS/2 CICS applies to CICS Operating System/2 (CICS for OS/2).
- Workstation applies to a personal computer, not an AIX workstation.
- The make process applies to the generic process not to specific make commands, such as make, nmake, pmake, polymake.
- Unless otherwise noted, references to VM apply to Virtual Machine/Enterprise Systems Architecture (VM/ESA) environments.
- References to VM batch apply to any batch facility running on VM.
- DB2/2 applies to DB2/2 Version 2.1 or later, and DB2 Universal Database (UDB) for OS/2 Version 5.
- DB2/6000 applies to DB2/6000 Version 2.1 or later, and DB2 Universal Database (UDB) for AIX Version 5.
- Windows applies to Windows 95, Windows 98, Windows NT, and Windows 2000.

- Unless a specific version of Windows NT is referenced, statements regarding Windows NT also apply to Windows 2000.

Terminology differences between Java and Smalltalk

VisualAge Generator Developer can be installed as a feature of VisualAge for Java or VisualAge Smalltalk. Where appropriate, the documentation uses terminology that is specific to Java or Smalltalk. But where the information is specific to VisualAge Generator and virtually the same for both environments, the Java/Smalltalk term is used.

Table 1. Terminology differences between Java and Smalltalk

Java term	Combined Java/Smalltalk term	Smalltalk term
Project	Project/Configuration map	Configuration map
Package	Package/Application	Application
Workspace	Workspace/Image	Image
Beans palette	Beans/Parts palette	Parts palette
Bean	Visual part or bean	Visual part
Repository	Repository/ENVY library	ENVY library manager
Options	Options/Preferences	Preferences

About this document

This document contains the following information:

- The program design methodology to consider when using VisualAge Generator
- Information on developing Structured Query Language (SQL) programs using VisualAge Generator
- Information on developing Data Language I (DL/I) programs using VisualAge Generator
- Information on allocating and associating files
- Considerations for the design and development of segmented, nonsegmented, and single-segment programs
- Considerations for the design and development of VisualAge Generator programs in the IMS/VS and IMS BMP environments
- Considerations for the design and development of VisualAge Generator programs in CICS environments
- Considerations for the design and development of VisualAge Generator programs in an OS/400 environment
- DBCS naming conventions, mixed data fields, testing and relational database support for programs containing DBCS
- Coding techniques that have significant performance or ease-of-use benefits
- The VisualAge Generator standards for data items and program libraries

Who should use this document

This document is intended for application system developers who will be developing programs for any of the environments supported by VisualAge Generator. The intent of this document is to help you plan an effective design before you begin coding your programs. Each chapter is written with a specific aspect of computer programming in mind.

Documentation provided with VisualAge Generator

VisualAge Generator documents are provided in one or more of the following formats:

- Printed and separately ordered using the individual form number.
- Online book files (.pdf) on the product CD-ROM. Adobe Acrobat Reader is used to view the manuals online and to print desired pages.

- HTML files (.htm) on the product CD-ROM and from the VisualAge Generator web page (**http://www.ibm.com/software/ad/visgen**).

The following books are shipped with the VisualAge Generator Developer CD. Updates are available from the VisualAge Generator Web page.
- *VisualAge Generator Getting Started* (GH23-0258-01) [1,2]
- *VisualAge Generator Installation Guide* (GH23-0257-01) [1,2]
- *Introducing VisualAge Generator Templates* (GH23-0272-01) [2,3]

The following books are shipped in PDF and HTML formats on the VisualAge Generator CD. Updates are available from the VisualAge Generator Web page. Selected books are available in print as indicated.
- *VisualAge Generator Client/Server Communications Guide* (SH23-0261-01)[1, 2]
- *VisualAge Generator Design Guide* (SH23-0264-00) [1]
- *VisualAge Generator Generation Guide* (SH23-0263-01) [1]
- *VisualAge Generator Messages and Problem Determination Guide* (GH23-0260-01) [1]
- *VisualAge Generator Programmer's Reference* (SH23-0262-01) [1]
- *VisualAge Generator Migration Guide* (SH23-0267-00) [1]
- *VisualAge Generator Server Guide for Workstation Platforms* (SH23-0266-01) [1,4]
- *VisualAge Generator System Development Guide* (SG24-5467-00) [2]
- *VisualAge Generator User's Guide* (SH23-0268-01) [1, 2]
- *VisualAge Generator Web Transaction Development Guide* (SH23-0281-00) [1]

The following documents are available in printed form for VisualAge Generator Server for AS/400 and VisualAge Generator Server for MVS, VSE, and VM:
- *VisualAge Generator Server Guide for AS/400* (SH23-0280-00) [2]
- *VisualAge Generator Server Guide for MVS, VSE, and VM* (SH23-0256-00) [2]

The following information is also available for VisualAge Generator:
- *VisualAge Generator External Source Format Reference* (SH23-0265-01)
- *Migrating Cross System Product Applications to VisualAge Generator* (SH23-0244-01)
- *VisualAge Generator Templates V4.5 Standard Functions—User's Guide* (SH23-0269-01)[2, 3]

---

1. These documents are available as HTML files and PDF files on the product CD.
2. These documents are available in hardcopy format.
3. These documents are available as PDF files on the product CD.
4. This document is included when you order the VisualAge Generator Server product CD.

Chapter 1. Designing VisualAge Generator programs

Note: In this chapter, the term *business application* refers to the end-user function being provided, such as Payroll or Inventory. The term *application system* refers to the set of GUI clients, VisualAge Generator programs, and/or non-VisualAge Generator programs that implement the function. These terms do not correspond to the grouping of parts into an ENVY package/application.

Before beginning design, you should know and understand the requirements of the application system you are developing. To develop systems that are modular, efficient, and easy to maintain, it is important to complete the initial design before you begin coding. VisualAge Generator supports and encourages an iterative development process.

The first step is to understand the business function required by the system's users; that is, the data to be processed and the operations that users want to perform with the data. If you understand the business function, you can more effectively describe the structure of the application system.

The basic functions found in most business applications are the following:
- Entry and update of data elements
- Comparison, analysis, and conversion of data elements
- Presentation of data on either a terminal or a printer

Therefore, the design tasks for an individual program include the following:
- Data design
- Logic design
- User interface design

An important design feature of an application system in VisualAge Generator is *modularity*. Every identifiable component in a program or GUI client is defined independently and stored in a Repository/ENVY library as a part. Each part is associated with other parts through definition references. Table 2 illustrates the VisualAge Generator part classes.

Table 2. VisualAge Generator Part Classes

Function	VisualAge Generator Part
Logic unit	A *function*
Data structure	A *record*, containing data elements

Table 2. VisualAge Generator Part Classes (continued)

Function	VisualAge Generator Part
User interface	A *map* (text or character-based user interface) or *view* (graphic user interface) that describes how data is presented to users
Set of values	A *table*, used for validation or conversion
Data element	A *data item*, referenced in records, maps, tables, functions
Transaction or batch program	Logic with associated maps and data

The VisualAge Generator approach to design includes independent design and definition tasks for data, user interfaces, and logic. The components are associated with one another through VisualAge Generator program definition and GUI client definition. Independent design and definition tasks allow you to concurrently associate each component with more than one program or GUI client.

The modular design approach of VisualAge Generator strongly supports developing client/server systems. With VisualAge Generator you can generate any client or server independently.

VisualAge Generator Developer can be used early in the design to create prototypes of your application system. By demonstrating a prototype system to your end-users, they can more easily verify that your design meets their requirements. After they verify your design, you can finalize the detailed data, logic, and operating system considerations for the application system.

When you have finished the initial design of your VisualAge Generator application system, use VisualAge Generator Developer to define and test your system. When your system is complete, use VisualAge Generator Developer to generate the final programs. For more information on defining and testing, refer to the VisualAge Generator help facility or to the *Programmer's Reference* document. For more information on generating your programs, refer to the *VisualAge Generator Generation Guide* document. For running your programs, refer to one of the following documents:

- *VisualAge Generator Server Guide for Workstation Platforms*
- *VisualAge Generator Server Guide for AS/400*
- *VisualAge Generator Server Guide for MVS, VSE, and VM*

Designing data

A first step in designing application systems is to define the data to be processed. This data can be defined in VisualAge Generator Developer independently, or as part of a structured set of entities (or objects). The entities can be associated with attributes and can have defined relationships with each other. The end-users are usually the best source for identifying this information.

Using an automated data modeling tool can help you capture and document the data descriptions. A tool that automatically transforms your data definitions to a physical database or file description enables you to design your programs more quickly and to maintain your programs longer. VisualAge Generator Developer enables you to develop programs that work with relational (SQL) and hierarchical (DL/I) databases. You can also develop VisualAge Generator programs that use indexed, relative, and serial file systems.

Designing entities

Entities are objects whose representations are processed by programs. Examples of entities are customers, orders, and parts in stock.

In VisualAge Generator Developer, entities are represented by records. Your data design must describe the following information for each entity your program uses:

- Name
- Description
- Attributes
- Relationships between entities

Designing entity attributes

Attributes are the characteristics associated with entities. Examples of attributes are customer name, order number, and number of parts in stock.

In VisualAge Generator Developer, attribute definitions are represented by data items. Your design must describe the following information for each attribute:

- Name
- Description
- Rules to determine valid values for the attribute
- Rules for how values are shown to program users
- Rules for deriving the attribute's value from other attributes, if the value is not stored in a file or database

To create attribute characteristics that are the same in all record definitions for that attribute, define the attributes as VisualAge Generator shared data items.

Designing entity relationships

Entities can be related to each other in defined ways, referred to as *entity relationships*. The following are examples of entity relationships: The order entity and the customer entity have a relationship in that an order is placed for a customer. Similarly, a part entity has a defined relationship with the order entity in that a part is one of the items listed on the order.

VisualAge Generator Developer entity relationships are usually implemented the same way you implement entity attributes. That is, they are implemented as data items in a record definition. For example, an order record includes a customer identification number as one of the data items. In this case, the customer identification number defines the relationship between the customer entity and the order entity. If you use DL/I databases, relationships can also be implemented as hierarchical parent/child relationships between records in the database.

To define entity relationships, your design must include the following information for each relationship. This information enables you to define the data item that establishes the relationship and the processing related to it:

- Name
- Description
- Related entities
- Rules for determining whether the relationship is valid

Designing user interfaces

An important step in application design is defining the user interface. The user interface can be a graphical user interface (GUI), a text or character-based user interface, or a printer map. Regardless of the user interface you use, there are two important aspects to user interface design:

- Function, or the specification of *what* the user wants to accomplish by using the interface
- Presentation, or the specification of *how* the interface physically appears to the user

When you design a user interface, avoid including too much information on a single window, terminal screen, or printer map. It is recommended that you use a separate window or terminal screen for each entity or array of entities to be processed. Encourage end-users to assist in identifying both the operations that apply to the entity presented on the interface and any navigation required between this user interface and other user interfaces. For example, if the user is taking an order from a new customer, link the interfaces together to enable the user to go directly from the order entry user interface to the customer entry user interface.

User interface function design

In developing the function design, you should work with the end-users to ensure your designs meet their requirements. Identify both the information they need to see or enter, and the operations they need to perform on the data. Examples of operations might be adding a new customer to a database, taking an order, and determining whether a part is in stock.

During the design of the logical user interface, it is a good time to determine which operations are initiated directly through the user interface, and which operations are started by a user action. An example of an operation initiated by the user interface is taking an order, and an example of an operation initiated by a user action is creating a purchase order for a part when inventory is decreased by an order.

User interface presentation design

With VisualAge Generator Developer, you can design two types of physical user interfaces:

- Graphical user interface (GUIs)
- A text or character-based user interface.

You must choose the interface appropriate for your runtime environment and for the needs of your end-users.

You can use VisualAge Generator Developer to prototype the physical user interfaces of your application system. Using prototypes, you can verify that the logical and physical user interface design meets your users' needs. With a prototype you can easily and quickly make changes to finalize your design.

Designing GUI interfaces

Consistency is very important when designing user interfaces. The VisualAge Generator GUI Composition Editor enables a great amount of flexibility in defining user interfaces; therefore, it is possible to make choices that can affect the consistency of your user interface.

To achieve consistency, you should implement a standard set of guidelines for all developers on a project to follow. There are many different user interface design guidelines available. The following are two recommended sources:

- *Object-Oriented Interface Design, IBM Common User Access (CUA) Guidelines* SC34-4399
- *OSF/Motif Style Guide, Revision 1.2 (For OSF/Motif Release 1.2)* (ISBN 0-13-643123-2)

Keep the following user interface design considerations in mind:

- Place the end-user in control

Text-based user interfaces force users to follow a strictly prescribed path of action, but graphical interfaces enable the user to follow many different paths to complete the same task. The following are guidelines to help you place the end-user in control:
  - Adopt the end-user's perspective
  - Keep interfaces flexible
  - Provide rapid response
- Reduce the amount of information the user must retain to effectively use the application

  You can reduce the amount of information the user must remember by making direct manipulation of screen objects possible. An example is enabling users to copy text from one source to another without having to remember data and type it again. Some guidelines to help you reduce the memory load are as follows:
  - Use progressive disclosure

    Progressive disclosure means you provide the information users need to perform their tasks as they need it.
  - Use real-world metaphors
  - Enable direct manipulation
- Reduce the end-user's error rate

  Consistency of interfaces increases the transfer of knowledge across application systems and the predictability of how systems act. Some guidelines to help you reduce the error rate are as follows:
  - Provide clues to natural progression of the task
  - Do not clutter the interface
  - Disable and enable controls as appropriate (for example, if two controls are mutually exclusive, one of them should be disabled at all times)

Some recommended methods for designing GUI interfaces are as follows:

- Use menu actions or push buttons to represent the operations that the user can perform or select. For each operation that an end-user can request, connect the menu actions or push buttons to the appropriate processing blocks. For navigation between visual parts, connect menu actions or push buttons to other visual parts.
- To display information and enable data entry, use graphical controls such as entry fields, tables, or radio buttons. Define the VisualAge Generator working storage records to pass information between clients and servers, connecting the graphical controls to the records.

For more detailed information on defining and implementing GUI clients, refer to the VisualAge Smalltalk documentation and the online VisualAge Generator help facility or the VisualAge for Java online VisualAge Generator help facility.

Designing logic

After you design the data, its entities, attributes and their relationships, and design the logical and physical user interface, you must design the structure that implements the operations.

An effective method of implementing designs is with client/server systems. In client/server systems, the user interfaces are implemented as text-based programs or graphical views, and the database and file operations are implemented as server programs. In this model, the clients request information and processing from the server programs.

A client/server design with VisualAge Generator has the following benefits:

- You can choose to generate and run your programs on a single system or on a network. On a network, the clients usually run on a workstation, while the server programs run on an OS/2, AIX, CICS for AIX, Windows NT, CICS for Windows NT, Solaris, CICS for Solaris, HP-UX, OS/400, CICS for MVS/ESA, CICS for VSE/ESA, or VM server.
- You can share a common set of server programs between GUI clients and text-based clients.
- You can change the target system for the client and server portions of the application system independently

Determining a client/server model

The model or architecture of your client/server application system affects how you design the individual programs and views in your system.

The location of the processing functions for your application system is an important consideration when deciding on a client/server model. The functional division determines where you create the boundaries of the components of your programs and views. The boundaries of your components directly affect the possibility of reusing your components. For more information on component reuse, see "Rapid application development through reuse" on page 15.

As you design, you should consider reusing the logic, I/O components, and parts of the user interface.

Client/server models

A client/server system consists of three logical components:

- Clients
- Servers
- Connecting network

In a client/server system, the client programs or views are separate from the server programs, and the clients request services from the servers. The logical

network that connects the clients and servers does not have to be a physical network. That is, the clients and servers can be on the same physical system.

Client and server data processing can be distributed using any of five different models. Figure 1 illustrates the five models.

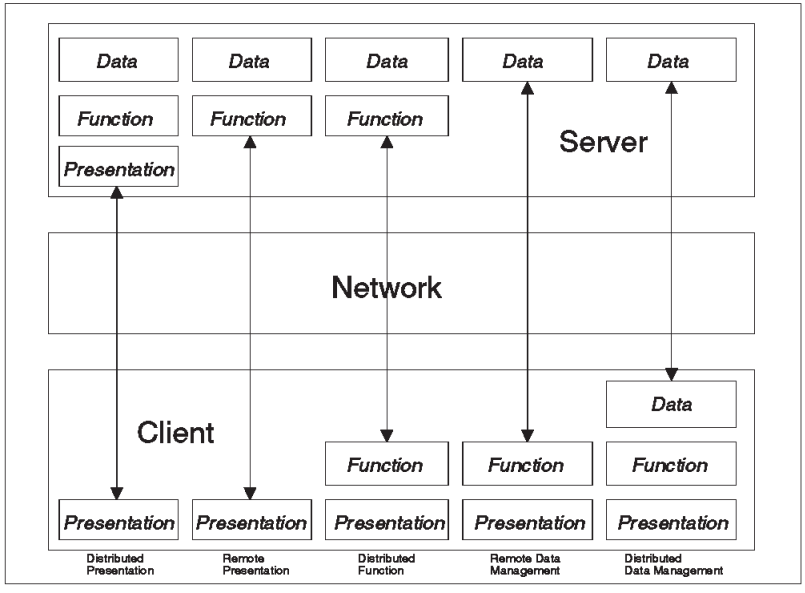

Figure 1. Five Client/Server models

Distributed presentation

In this model, the presentation, function, and data components are on the server platform and part of the presentation is on the client platform.

Distributed presentation is accomplished by using terminal emulation software like that provided with the IBM OS/2 Communications Manager. With this software, the user sees the VisualAge Generator host program map displayed on the workstation rather than on a 3270 or 5250 device, although the program is running on the host system instead of the workstation.

Remote presentation

In this model, the presentation component is located on the client platform while all the functions and data are located on the server.

Remote presentation is accomplished by collecting user input using either a text-based or graphical client, and then calling a server program to process the data. The user input is validated by the server program.

Distributed function

In this model, the presentation component is located on the client platform while the function is located on both the server and the client.

Of the five client/server models, the distributed function model is the most flexible, and can be the most efficient. The distributed function model enables you to grow and change your application system as your business needs change. You also have the option of placing your function and data components on the most cost-efficient platform.

Remote data management

In this model, the presentation and function are on the client platform and the data is on the server platform.

An example of remote data management is the Remote Unit of Work (RUW) in the Distributed Relational Database Architecture (DRDA). The RUW of DRDA is a relational database product feature. VisualAge Generator supports the RUW of DRDA in both developing and running programs. If you want to use RUW, you must determine whether or not your database manager supports DRDA with a RUW feature.

Another example of remote data management is the transaction manager function, *function shipping*, in the CICS family of products. Function shipping is the ability of one program running in a CICS system to gain access to different connected CICS systems. For example, CICS for OS/2 can gain access to data from CICS/ESA.

VisualAge Generator supports CICS function shipping in both developing and running programs.

Distributed data management

In this model, the presentation and function are located on the client, while portions of the data function are located on both the client and server.

An example of distributed data management is the Distributed Unit of Work (DUW) in Distributed Relational Database Architecture (DRDA). The DUW of DRDA is a database feature available in Version 2 of workstation DB2 products and is supported in VisualAge Generator.

VisualAge Generator supports valid DRDA connections for both developing and running programs.

Summary of client/server models

The distributed function model is the primary strength of VisualAge Generator client/server support; however, VisualAge Generator supports the four other client/server models under specific conditions and considerations. Some examples of these considerations or conditions are as follows:

- The flexibility and efficiency provided in the remote presentation and distributed function models has a cost: a complex, underlying communications infrastructure. A benefit of VisualAge Generator is the client/server communications support provided with the product. This communications support enables you to create programs in the remote presentation and distributed function models without needing to know the complex communications protocols that are used.
- VisualAge Generator supports the remote data management and the distributed data management models only through functions provided by database and transaction management products. The scope of the VisualAge Generator client/server support of these models depends on the capabilities of the database and the transaction management products used.
- To use the remote presentation model, the GUI client can only use visual connections to start server programs or other callable functions. That is, there can be no connections to functions on the Composition Editor's free-form surface (otherwise the model you are using is the distributed function model).
- VisualAge Generator only supports distributed presentation for a text-based user interface through terminal emulation tools.

For more information on client/server applications, refer to the *VisualAge Generator Client/Server Communications Guide* document.

Designing clients

A client should have a hierarchical relationship with the server program it calls for database or file access. Clients should either call, transfer, or asynchronously start other clients or servers.

The following is a list of considerations to keep in mind when designing clients:

- Generally, it is best to keep the design of individual clients as simple as possible. Consider having only one primary window or screen per view or program. Ensure that your user can easily identify the specific entity to be processed and easily select the operations to be performed on that entity.
- Your design should include a short summary of the function of each client program or view, and a detailed description of any information passed between them. Validate that the client formats the data displayed to the user correctly and that the input data received from the client is correct for the defined data.
- Minimize the number of calls and the volume of data flow between clients and servers when designing call interfaces. For example, design the server to return an array of entities instead of one entity at a time, or design the server program to return a calculated value instead of data used to perform a calculation.

- The design of the navigation, or flow of control, is determined by whether the interface is graphical or text-based. See "Designing GUI clients" on page 12 and "Designing text-based user interfaces" for more information on navigation design.

  For GUI and text-based interfaces, the design of the application system should maintain the current contents of variable fields used in a current working storage record. This working storage record is shared or passed among the clients. With this working storage record, you can set the calling view or program back to its original contents when control is returned to it.

Designing server programs

A server program should have a hierarchical relationship with the client that calls it for database and file access.

The following is a list of considerations to keep in mind when designing server programs:

- Generally, it is best to keep the design of individual server programs as simple as possible. Consider having only one server program per transaction or a closely related set of transactions.

  A transaction is a set of related database service requests. For database updates, a transaction represents a unit of work. The unit of work can be updates that must be synchronized to ensure the database is correct. For example, assume an order is committed to the database. An order transaction should update both the order and the inventory data records.

  Some environments support logical units of work, which can include multiple server calls. Other environments require each server call to be a separate unit of work. By putting all update transactions in a single server call ensures your design works in all supported operating environments.
- In VisualAge Generator, a server program is implemented as a *called batch program*. Your design should specify the program inputs and outputs as record structures. Describe internal logic flow as a hierarchy of procedural logic structure; that is, higher-level processing logic calling to lower-level processes to perform specific functions and then returning back to the higher-level process.
- Your design should include a short description of the processing logic implemented at each node in the structure. Assume only one I/O operation is performed in each process of the hierarchy. Implement the logic as VisualAge Generator functions. Use WHILE statements for logic that repeats (loops) and IF statements for conditional processing.

Designing text-based user interfaces

With VisualAge Generator Developer, you can define and generate text-based programs that can run independently can be used as clients to local or remote server programs.

When designing text-based programs, specify *Main Transaction* program. Use function keys or map variable values to represent operations that the user can select. Use a CONVERSE I/O option for the main function to display a map to the user. Define VisualAge Generator processing blocks (functions or other programs) for each operation that the program user can request. Use IF statements following the CONVERSE I/O options to CALL, or invoke the appropriate processing block based on which key was pressed. Define VisualAge Generator working storage records to pass information between clients and server programs.

To implement navigation between text-based programs, use DXFR statements. If a return to the current program is possible, save the state of the current program in the primary working storage record or in a file, and restore the state of the current program after program control returns from the server.

To display information and enable data entry, use variable fields on maps. Before using the CONVERSE I/O option, use MOVE statements to pass the information from working storage records to the map. To pass the information back to working storage from the map, use MOVE statements after the CONVERSE I/O option.

Designing GUI clients

GUI clients are *event-driven*. This means that they respond to user input by running logic, invoking other views or programs, or altering the style of the current view when the user interacts with it. Examples of ways users provide input are by clicking on a push button or entering text in a data entry field.

Designing GUI clients is different from designing text-based programs. The main differences are in the following:

- How you define the appearance of the user interface, especially with regard to how the functions are controlled
- How you design the logic

In the runtime environment of an event-driven GUI client, the view is supported by a variety of interface events, such as windows being opened and push buttons being clicked with a mouse.

When designing an event-driven GUI, consider the possible events that can take place and the order in which they can occur. You can prototype these events and design your view iteratively. Designing a prototype first can help you determine why, how, and what procedural logic should run, ensuring that the system response is predictable.

GUI clients strongly support client/server computing systems. VisualAge Generator Developer provides the ability to visually construct graphical user

interfaces that link selected objects or controls on a GUI to logic that can run either locally, on a remote system, or both.

At run time, the clients and servers can be located on the same system or on different systems.

GUI clients often must communicate with server programs. Communication requests to servers by GUI clients can be designed in two ways:

- Using CALL statements in VisualAge Generator functions to call server programs
- Connecting an event to a Callable Function part on the *free-form surface.*
- Coding the invocation of a Callable Function part on the *free-form surface* within a Smalltalk script or Java code

For more information on developing event-driven GUI clients, refer to the VisualAge Smalltalk documentation and the online VisualAge Generator help facility or the VisualAge for Java online VisualAge Generator help facility.

Performance considerations for GUI clients

The design of your GUI client can have significant effects on its runtime performance. Following are some design and coding techniques to improve the performance of your GUI clients when using VisualAge Generator parts and VisualAge Generator extensions to VisualAge Smalltalk or VisualAge for Java parts.

- Use *getFieldsStartingAt:to:* to populate Container Details parts

  When populating a Container Details part, use the *getFieldsStartingAt:to:* action from the tear-off attribute of the occurs item (array) in the VisualAge Generator record. Refer to "Handling Occurs Items" in the *Programmer's Reference* for more information.

  The *getFieldsStartingAt:to:* action of the torn-off occurs data item is more efficient than connecting directly to the occurs items. The *getFieldsStartingAt:to:* action copies from n to nn items of the array. By contrast, if the *items* attribute of the occurs item is connected to the *items* attribute of the Container Details part, the entire occurs item (array) is scanned, and all items up to and including the last non-default item are copied to the Container Details.

  For example, in an array with an occurs value of 50, if you connect the *items* to the Container Details, VisualAge Generator looks at each data item in the array to see if it is a non-default value before displaying the values. If the *getFieldsStartingAt:to:* action is used, VisualAge Generator copies the 50 items without the overhead of checking for a default value. Zeroes or blanks will show in this case. If the application can determine the number of valid data items in the array, then you can set parameters for the *getFieldsStartingAt:to:* action, further accelerating the Container Details population process.

An event needs to trigger the *getFieldsStartingAt:to:* action. Often the best event is the *hasExecuted;* of the VAGen Logic part in which the data for the Container Details is accumulated or created. Then define the parameters as appropriate in the **Settings** or with connections to the parameters on the event-to-action connection. Finally, connect the *items* of the Container Details to the result of the event-to-action connection.

- Use *getValuesStartingAt:to:* to populate List parts

  When populating a List part, use the *getValuesStartingAt:to:* action in the same way as described above for the *getFieldsStartingAt:to:* action for Container Details parts. Values are used instead of fields because Lists are usually populated with simple arrays, while Container Details parts are connected to a compound array with one or more substructures. Thus, use fields to assure that values are mapped to the correct fields or columns.

- Minimize connections to information shared between visual or non-visual parts.

  You can share data of VAGen data parts between parts. However, you should minimize the number of other connections to that shared information. When one visual part opens another, all the attribute connections between the parts are triggered. If the shared information also has connections to other information, then those connections are also triggered, creating a chain reaction. This can happen when another visual part is opened or when the data changes. Use unidirectional attribute-to-attribute connections where appropriate to reduce the amount of triggering.

- Use unidirectional connections when appropriate

  You should use unidirectional attribute-to-attribute connections whenever possible. It is also a good practice to remove any unnecessary attribute-to-attribute connections to eliminate unneeded data movement. Note that using **Quick Form** creates default bi-directional attribute-to-attribute connections. You should evaluate these connections to see if they can be changed to unidirectional connections or removed.

- Limit the number of occurs in the record used as a parameter for the VAGen *performRequest* action.

  When searching an occurs item that is a parameter for a VAGen *performRequest* action, VisualAge Generator searches all occurs values to determine if an action is to take place. The search does not stop at the first blank action. Therefore, the structure should be as small as possible to avoid the overhead of searching through blank actions.

- Minimize the amount of data passed to server programs

  It is important to minimize the amount of data passed across the network to reduce network traffic. If data conversion is done, each byte is converted as it goes from client to server and back, even if it is a blank.

Minimize the use of substructured data items on a call that goes across the network. There is overhead when passing substructures across the network.

- Minimize the use of decimal places for numeric data

  VisualAge Generator GUI client runtime optimizes data items with no decimal positions.

- Use level-77 data items whenever possible

  Because level-77 data items are not part of the record structure, it is more efficient to use them in the GUI client logic. For example, you should use level-77 data items for loop counters.

- Minimize the use of substructured records and occurs items. When possible, use flat record structures when passing to Callable Functions or sharing with other GUI client parts.

  GUI client runtime is not optimized for substructured records or for records with occurs item.

- Minimize connecting the *data* attribute of records

  GUI client runtime is not optimized for records that have connections to or from the record's *data* attribute. Instead, use the VAGen Variable part.

  For more information on the VAGen Variable part, refer to the *Programmer's Reference*

- Do not place records or other VAGen data or logic parts on the Free-form surface that are not used by the GUI client.

  The structure for every record on the Free-form surface is generated in the VisualAge Generator runtime code of the client part. Therefore, unnecessary records increase the amount of storage used at runtime.

- Minimize the movement of data between character and numeric data items. Movement between these data types requires type conversion, which is not optimal.

- Minimize the use of automatic truncation and padding. Although VisualAge Generator truncates or pads when the source and target of MOVE and ASSIGNMENT statements are different lengths, this requires extra code.

- Use the NONUMOVFL generation option whenever possible.

  Performance of GUI clients is significantly faster when the generated VisualAge Generator logic does not have to check for numeric overflow.

Rapid application development through reuse

The basis of reuse is to take an existing program, one with logic similar to what is needed for a new program, and generalize it to meet the needs of the new or existing programs. Reuse is an important design consideration. The following is a list of ways you can take advantage of reusing existing code:

- Create standard components and parts for common functions, such as handling database and file errors. Standard components can be directly included in many different programs or views.
- Create model or template programs for common operations, such as scrolling in database tables and re-establishing an update position in a database following user interface input. While not fully reusable as is, these components can be copied and modified for each specific usage. This is known as cloning.

Reusable components can simplify your design. For example, you can state in the design that a reusable component is used for an operation instead of repeating the description.

Contact your IBM representative for service offerings and products that provide reusable sample programs and a methodology for using and updating reusable components.

Program size considerations

As you design new VisualAge Generator application systems, design the system as a set of small (1500 statements or less) modular programs instead of as one very large program. Smaller programs are more efficient to test and maintain.

If your target runtime environment is a workstation system, it is usually best to keep your VisualAge Generator program under 1500 VisualAge Generator statements, in order to avoid long compile times.

The VisualAge Generator Developer produces at least twice as many COBOL source statements as the number of original VisualAge Generator source statements. The actual size of the generated COBOL source depends on the size of the original VisualAge Generator source, the type of VisualAge Generator statements used in the program, the amount of function they provide, and the complexity of the VisualAge Generator program. An exact correlation between the size of the generated COBOL program and the VisualAge Generator source is not possible, but the generated COBOL source is always much larger.

Some examples of the greater productivity achieved from generating VisualAge Generator COBOL, C++, Smalltalk, or Java versus handwritten programs are the following:

- A single procedural VisualAge Generator statement such as MOVE automatically handles operands of unequal lengths, validates data types, and performs conversion as necessary. A single VisualAge Generator

statement also sets SQL indicators for SQL data items, handles repeated moves if the target of a move is an array, and handles overflow if required as a generation option.

- VisualAge Generator procedural statements such as IF..IN and RETRIEVE/FIND provide built-in functions that otherwise require additional logic to be hardcoded.
- Program logic is generated to handle pseudoconversational processing for CICS environments.
- File I/O options are transformed into source code to open files, invoke runtime services routines to keep track of file use count, issue the file read, handle the return code and transform it into VisualAge Generator mnemonics, and invoke EZERTN file error routine if necessary.

Chapter 2. Developing SQL programs

This section describes how to develop Structured Query Language (SQL) programs using VisualAge Generator. You can quickly develop SQL programs without coding SQL statements because VisualAge Generator creates the SQL statements for you.

You can use SQL to access information stored in relational databases. Table 3 outlines the database manager that supports SQL access to IBM relational databases for each environment. The database products differ in minor respects.

Table 3. IBM Database Managers used in each Environment

Database Manager	Environment
DATABASE 2 (DB2)	MVS systems
DB2/VSE (formerly SQL/DS)	VSE systems
SQL/DS	VM systems
DB2/400	OS/400 systems
DB2/2	OS/2 systems
DB2/6000	AIX systems
DB2 for Windows NT	Windows NT systems
DB2 for HP-UX	HP-UX systems
DB2 for Solaris	Solaris systems

If you are testing programs with one database manager and generating the program to run with a different database manager, you should be familiar with the publications for both database managers. Use your SQL documentation when modifying the default SQL statements. Some of the differences relate to handling long character data types and resource recovery. The VisualAge Generator test facility and the generation facility mask most differences when default SQL statements are used in the program.

The information in this chapter applies to both IBM and non-IBM database management systems supported by VisualAge Generator unless otherwise noted. For differences and more specific information on accessing non-IBM databases, see "Accessing databases using DataJoiner" on page 67, "Accessing databases using ODBC" on page 69 or "Accessing databases using Oracle" on page 74.

Understanding SQL and relational databases

Before you begin developing an SQL program, you need to understand the relationship between VisualAge Generator and the following basic SQL concepts:

Tables All data in a relational database is represented as two-dimensional tables with columns and rows. Tables in a relational database are similar to other tables you use every day, such as phone directories, airline schedules, and stock market reports.

Tables are defined to the database manager using the CREATE TABLE SQL statement. The CREATE TABLE statement defines the name of each column in the table, and the data type and length of the values that appear in the column.

You can define a relational table to VisualAge Generator as a record with SQL row organization. You can use the record definition facility to define SQL row records. The SQL record definition includes the following information:

- Names of the tables in the database that the record represents; or table name host variables that the program moves the actual table name into at run time
- Items in the record that represent columns in the tables
- Default selection criteria used in generating SQL statements that access the tables

Operations on Tables

After a table is defined to the database manager, rows of data can be added to it. Each row contains one data value for each column defined in the table. Rows in the table can be retrieved, updated, inserted, or deleted using SQL statements from a system command line, a terminal, or a program. From a system command line or a terminal, you can search several rows in a table at the same time. In a program, you can process the data in a table one row at a time.

Defining SQL programs is similar to defining other VisualAge Generator programs. From I/O options, VisualAge Generator creates default SQL statements that access the tables in a relational database. You can modify some of the statements to request additional SQL functions. If you are working with a single table that has a column that can be used as a unique key, you can process the table without modifying SQL statements because VisualAge Generator defines the necessary SQL statements for you by default. The following I/O options are available for use with SQL row records:

INQUIRY

Get a row from a table

UPDATE
Get a row from a table for replacement or deletion

ADD Insert a row in a table

REPLACE
Replace a row in a table that was read using an UPDATE or SCAN function

DELETE
Delete a row from a table that was read using an UPDATE or SCAN function

SETINQ
Select a set of rows from a table for scanning

SETUPD
Select a set of rows from a table for scanning with possible replacement or deletion

SCAN Get the next row from a set of rows that were selected using a SETINQ or SETUPD function

CLOSE
End the scanning of a selected set of rows before the last row has been retrieved

SQLEXEC
Run a user-defined SQL statement

Views A view is a logical (or virtual) table that is derived from one or more tables or views or combinations of tables and views. To a program, a view looks like a table. It has a name and contains data arranged in columns and rows. You can use a view name anywhere you use a table name in defining records and logic for SQL programs.

Joins A join is a relational database operation that creates a logical (or virtual) table by forming combinations of rows from two or more tables. Usually some conditions are defined to limit the number of combinations. For example, in a relational database for a college, a faculty table might be joined with a course offering table. The resulting joined table includes only the combinations of rows for a professor who taught a course.

In VisualAge Generator Developer record definition, you can specify that an SQL row record represents a join of two or more tables, and you can also enter the conditions that control the combinations of rows included in the join.

Null Data
Any field in a relational database table might have a *null* value. Null represents an unassigned or undefined value. A null value can be

thought of as an empty space or space reserved for later insertion of data. Nulls can be set in a column through insert or update operations. Nulls can also occur implicitly when a new column is added to an existing table. Null values can be prohibited for specific columns when a table is created.

You can design your program to test or set null values for items in SQL row records.

If you are using relational databases for the first time, it is a good idea to completely understand what you can do with SQL by experimenting with the appropriate interactive facility. For DB2, you can use one of the following:

- The Interactive SQL (ISQL) facility of DB2/VSE
- The Start SQL (STRSQL) facility of DB2/400
- The Start Query Manager (STRQM) of DB2/400
- The DB2 Interactive (DB2I) facility of DB2
- The Query Manager, Command Line Processor, Command Center (DB2 Version 5 or later) and online help of DB2/2 or DB2 for Windows NT
- IBM Visualizer Query for AIX
- The Interactive SQL (ISQL) facility, and the Database Services Utility for SQL/DS

See your database administrator for more information about relational database concepts and facilities available at your installation.

Example: Inventory, Suppliers, and Quotations Sample Tables

The examples in this section show the sample tables shipped with the DB2/VSE program product. The Inventory, Suppliers, and Quotations sample tables are used throughout this chapter to illustrate the coding of a VisualAge Generator SQL program. The tables contain information needed for maintaining inventory, such as the part descriptions, part numbers and the quantity in stock, the part supplier's names and addresses, each supplier's price, delivery time, and quantity on order for parts. The tables are defined to VisualAge Generator as SQL row records.

The columns in the tables are defined in the example SQL database as shown in Table 4:

Table 4. Example SQL Database

Name	SQL Data Type
PARTNO	SMALLINT (NOT NULL)
DESCRIPTION	VARCHAR(24)
QONHAND	INTEGER
SUPPNO	SMALLINT (NOT NULL)
NAME	CHAR(15)
ADDRESS	VARCHAR(35)
PRICE	DECIMAL(5,2)

Table 4. Example SQL Database (continued)

Name	SQL Data Type
DELIVERY_TIME	SMALLINT
QONORDER	INTEGER

Inventory Table

Table 5 contains information on parts in stock: part number (PARTNO), part description (DESCRIPTION), and quantity on hand (QONHAND).

Table 5. Inventory

PARTNO	DESCRIPTION	QONHAND
207	gear	75
209	cam	50
221	bolt	650
222	bolt	1250
231	nut	700
232	nut	1100
241	washer	6000
285	wheel	350
295	belt	85

Suppliers Table

Table 6 contains information about suppliers: supplier number (SUPPNO), name (NAME), and address (ADDRESS).

Table 6. Suppliers

SUPPNO	NAME	ADDRESS
52	Vesuvius, Inc.	512 Ancient Boulevard, Pompeii, New York
53	Atlantis Co.	8 Ocean Avenue, Washington, D.C.
54	Titanic Parts	32 Large Street, Bigtown, Texas
57	Eagle Hardware	64 Tranquility Place, Apollo, Minnesota
61	Sky Parts	128 Orbit Boulevard, Sydney, Australia
64	Knight Ltd.	256 Arthur Court, Camelot, England

Quotations Table

Table 7 contains information about quotations from suppliers for specific parts: supplier number (SUPPNO), part number (PARTNO), price (PRICE), delivery time (DELIVERY_TIME), and quantity on order (QONORDER):

Table 7. Quotations

SUPPNO	PARTNO	PRICE	DELIVERY_TIME	QONORDER
51	221	.30	10	50
51	231	.10	10	0
53	222	.25	15	0
53	232	.10	15	200

Table 7. Quotations (continued)

SUPPNO	PARTNO	PRICE	DELIVERY_TIME	QONORDER
53	241	.08	15	0
54	209	18.00	21	0
54	221	.10	30	150
54	231	.04	30	200
54	241	.02	30	200
57	285	21.00	14	0
57	295	8.50	21	24
61	221	.20	21	0
61	222	.20	21	200
61	241	.05	21	0
64	207	29.00	14	20
64	209	19.50	7	7

Defining relational database tables

A relational table is defined to VisualAge Generator as an SQL row record. The record can represent one table or table view, or a join of two or more tables or table views. Unless otherwise noted, information about single tables also applies to joined tables.

Defining relational tables as record parts

You use VisualAge Generator Developer record definition to define a relational table as an SQL row record part in the Repository/ENVY library. You can name the record part anything you want. The part name does not have to be the same as the table name.

An SQL row record definition can represent a join of two or more tables or views.

The only I/O options that can be used with a join are functions that read the database, such as:

- INQUIRY
- SETINQ
- SCAN

Defining data items for SQL row records

For SQL row records, the data items defined represent columns in a relational table instead of a record structure. In addition to the item name, specify the column name for the item as it is known to the database manager. You cannot specify level and an OCCURS value for any item in an SQL row record.

When you define an SQL row record on a system with access to a relational database, you can retrieve and display the SQL column information from the database. If a database is not specified, or if the table is in host variable

format, the Retrieve and Compare options are not available, and the SQL data types are converted to VisualAge Generator data types. If the database is not available, the relational table is not defined, or if the table is in host variable format, you have to define the data items.

For shared data items, the item name, type, length, decimal places, bytes, and description values are all stored with the data item part in the Repository/ENVY library. For nonshared data items, the item name, type, length, decimal places, bytes, and description values are all stored with the SQL row record in the Repository/ENVY library. The SQL data code values are associated with the item as it appears in the SQL row definition and are stored with the SQL row, not with the data item part.

Note: When VisualAge Generator Developer obtains the SQL definitions for an SQL row record during record definition, it uses the unqualified SQL column names as the default list of data item names. VisualAge Generator Developer does not truncate column names, but if the item name is not a valid VisualAge Generator name or if it duplicates the name of an item in another structure, you receive an error message. You must correct the errors before saving the definition.

Modifying the data item definition

You might modify a data item definition if you are following a naming convention that the SQL columns does not use. You can modify the data item definition extracted from the relational database as follows:

- Change item names.

  VisualAge Generator uses the SQL column name associated with the item when it creates the SQL statements that access the database. The item name is used to identify the item within the program definition.
- Identify default selection columns.

  On the SQL Row Item Usage window, select the **Key Item** check box for items to be used as search columns in the default SQL statements that are created for the record. The use of keys affects the creation of SQL row default statements. See "Defining default selection conditions" on page 26 for additional information on key items.
- Arrange the items in an SQL row.

  The order you specify data items as keys determines the ORDER BY clause for the SETINQ I/O option.
- On the SQL Row Item Usage window, select the **Read-only** check box for items your program does not modify.

  You should select the **Read-only** check box for items that represent virtual columns in views that cannot be written back to the database.
- Delete columns from an SQL row that your program does not need to process.

- Change the item data type and length from the type and length obtained from the SQL catalog.

  The VisualAge Generator and SQL data types must be compatible.
- Add items whose SQL column name is defined as an expression with the actual column name used as one of the operands in the expression.

Comparing the SQL row definition to the database definition

You can compare your SQL row record definition to the definition of the tables in the database when the following conditions exist:

- The database is available
- Your relational table exists
- You do not have any table name host variables

If a database is not specified, or if the table is in host variable format, the Retrieve and Compare options are not available. If the relational database is available, the record definition function extracts and uses the table definition from the relational database. The record definition function tries to match each column in the relational table with a column name specified in the record. The data definition function then displays each column that does not have a matching data item, each data item that does not have a matching column, and all data items having an SQL column definition that is different from the Repository/ENVY library data item definition.

Defining default selection conditions

In a SELECT SQL statement, the WHERE clause identifies the criteria used to select rows from the database. You can control the default WHERE clauses that VisualAge Generator creates for functions that access the SQL row record. You control the default WHERE clause by specifying key items and default selection conditions:

Key items

You can identify data items in the SQL row record as key items in the SQL row record definition. For alternative specification records, you can specify a single default key item during record definition.

For single-record I/O options, such as INQUIRY and UPDATE, VisualAge Generator uses items to create the default WHERE clause to select records that match the values of the data items specified as keys.

For multi-record I/O options, such as SETINQ and SETUPD, VisualAge Generator generates key selection clauses only if a single key is specified. The records selected have a key column value that is greater than or equal to the values of the data item used as the keys.

Keys are sorted in the default ORDER BY clause in the order you defined the key items. Key items are excluded from the FOR UPDATE

OF clause and the SET clause to maintain referential integrity. See "Referential integrity considerations" on page 57 for general information on referential integrity.

Default selection conditions

You enter the conditions in a WHERE clause of a simulated SELECT statement. Use default selection conditions to specify join conditions when the record represents a join of two or more tables.

During function definition you can view the default WHERE clause created for a function. If you specified both key items and selection conditions, the default WHERE clause combines the key search and the selection conditions using an AND operator. For SETINQ or SETUPD I/O options on multiple key records, default selection conditions are defined; however, a default WHERE clause is not defined.

If the default WHERE clause does not meet the requirements for a function, you can modify the clause for any individual function.

Joined Tables

Often when you define a record as a join, you also want to define selection conditions that limit the number of rows used from the combined tables.

In high-level language programs, you must define the join conditions in each SELECT statement that accesses the join tables. With VisualAge Generator, after you define the join conditions with the SQL row record definition, the join conditions are automatically included in any SELECT statement created for the SQL row.

SQL data type support

The SQL data type identifies the data type of the data item in the relational database. VisualAge Generator uses DB2 SQL data types to map to VisualAge Generator data types for SQL row records. SQL data types for other database management systems are mapped to DB2 data types during SQL column retrieval and comparison. SQL data codes can vary only for double-byte character set (DBCS) characters and hexadecimal data items. SQL data codes are fixed for other types of data items and cannot be modified.

Variable length and fixed length items

You can specify that the table column associated with an item in an SQL row record is variable or fixed length by setting the SQL data code for the item to the following values:

Table 8. SQL Data Types for Variable and Fixed Length Columns

VisualAge Generator Data Type	SQL Data Type	Variable/Fixed
CHA	453–CHAR (default)	Fixed
CHA	449–VARCHAR, length < 255	Variable
CHA	457–VARCHAR, length > 254	Variable
DBCS	469–GRAPHIC (default)	Fixed
DBCS	465–VARGRAPHIC, length < 128	Variable
DBCS	473–VARGRAPHIC, length > 127	Variable

Note: The SQL data types associated with record data items always specify the use of null indicators. This does not affect the table column associated with the data item, which might be defined as null or not null. See "SQL Nulls" on page 34 for a description of the use of null values in generated programs.

The SQL data type for a variable length column is saved automatically for a data item whenever the relational table definition is defined from the SQL catalog. You can also set the SQL data code using the SQL Row Item Usage window for SQL row records.

If an item is associated with a variable length column, the column is accessed using a variable length host variable whenever the column is read from the table or written to the table.

Compatible data types

SQL provides the necessary conversion between compatible data types. A data item in an SQL row record and its corresponding column in the SQL database are considered compatible if one or more of the following is true:

- The SQL column is any form of character data, and the data item is of type CHA with length less than or equal to the length of the SQL column.

  Data read from a column into a shorter data item is truncated on the right. You can test for truncation using the IF item TRUNC statement.
- The SQL column is any form of DBCS data, and the data item is of type DBCS with a byte length less than or equal to the length of the SQL column.

  Data read from a column into a shorter data item is truncated on the right. You can test for truncation using the IF item TRUNC statement.
- The SQL column is any form of number and the data item is the following:
  - Binary (BIN) 2 or 4 bytes, no decimal places

– Packed (PACK) with a maximum length of 18 digits, including decimal places

Note: C++ does not have a packed decimal data type. Floating point variables with a maximum precision of 15 digits are used to access decimal columns. SQL columns of up to 18 digits can be accessed, but the numbers will be accurate only to the first 15 significant digits.

If data is read from a number column into a shorter data item, leading zeros are truncated on the left. If the number still does not fit into the data item, fractional parts of a decimal number are deleted on the right with no error indication. If the number still does not fit, an overflow condition results and a negative SQL code is returned.

- The SQL column has any type, the data item has type HEX, and the column and item have the same length. No data conversion is performed when data is transferred. HEX data items support access to SQL columns whose SQL data type does not have a corresponding VisualAge Generator data type.

SQL-to-VisualAge Generator data type conversion

When a table definition is extracted from the database, the SQL data types are converted to VisualAge Generator data types as shown in Table 9. The table shows the DB2/400, DB2/VSE, SQL/DS, DB2, DB2/2, DB2/6000, DB2 for Windows NT, DB2 for HP-UX and DB2 for Solaris data types and the corresponding VisualAge Generator data types.

Table 9. SQL to VisualAge Generator Data Type Conversion

SQL Data Type	DB2/400	DB2/VSE or SQL/DS Type	DB2/MVS Type	DB2/2, DB2/6000, DB2 for Windows NT, DB2 for HP-UX, DB2 for Solaris Type
CHAR	Length: 1–32766 Type: CHA Bytes: 1–32766	Length: 1–254 Type: CHA Bytes: 1–254	Length: 1–254 Type: CHA Bytes: 1–254	Length: 1–254 Type: CHA Bytes: 1–254
VARCHAR	Length: 1–32740 Type: CHA Bytes: 1–32740	Length: 1–254 Type: CHA Bytes: 1–254	Length: 1–32767 Type: CHA Bytes: 1–32767	Length: 1–4000 Type: CHA Bytes: 1–4000

Table 9. SQL to VisualAge Generator Data Type Conversion (continued)

SQL Data Type	DB2/400	DB2/VSE or SQL/DS Type	DB2/MVS Type	DB2/2, DB2/6000, DB2 for Windows NT, DB2 for HP-UX, DB2 for Solaris Type
LONG VARCHAR	N/A	Length: 32767 Type: CHA Bytes: 32767	Length: 1–32767 Type: CHA Bytes: 1–32767	Length: 1–32700 Type: CHA Bytes: 1–32700
GRAPHIC	Length: 1–16383 Type: DBCS Bytes: 2–32766	Length: 1–127 Type: DBCS Bytes: 2–254	Length: 1–127 Type: DBCS Bytes: 2–254	Length: 1–127 Type: DBCS Bytes: 2–254
VARGRAPHIC	Length: 1–16370 Type: DBCS Bytes: 2–32740	Length: 1–127 Type: DBCS Bytes: 2–254	Length: 1–16383 Type: DBCS Bytes: 2–32766	Length: 1–2000 Type: DBCS Bytes: 2–4000
LONG VARGRAPHIC	N/A	Length: 16383 Type: DBCS Bytes: 32766	N/A	Length: 1–16350 Type: DBCS Bytes: 2–32700
SMALLINT	Length: 2 Type: BIN Bytes: 2	Length: 2 Type: BIN Bytes: 2	Length: 2 Type: BIN Bytes: 2	Length: 2 Type: BIN Bytes: 2
INTEGER	Length: 4 Type: BIN Bytes: 4	Length: 4 Type: BIN Bytes: 4	Length: 4 Type: BIN Bytes: 4	Length: 4 Type: BIN Bytes: 4
DECIMAL[1]	Length: 18 Type: PACK Bytes: 1-10	Length: 18 Type: PACK Bytes: 1-10	Length: 18 Type: PACK Bytes: 1-10	Length: 18 Type: PACK Bytes: 1-10
FLOAT	Length: 8 Type: HEX Bytes: 8	Length: 8 Type: HEX Bytes: 8	Length: 8 Type: HEX Bytes: 8	Length: 8 Type: HEX Bytes: 8
DATE	Length: 10 Type: CHA Bytes: 10	Length: 10 Type: CHA Bytes: 10	Length: 10 Type: CHA Bytes: 10	Length: 10 Type: CHA Bytes: 10

Table 9. SQL to VisualAge Generator Data Type Conversion (continued)

SQL Data Type	DB2/400	DB2/VSE or SQL/DS Type	DB2/MVS Type	DB2/2, DB2/6000, DB2 for Windows NT, DB2 for HP-UX, DB2 for Solaris Type
TIME	Length: 8 Type: CHA Bytes: 8	Length: 8 Type: CHA Bytes: 8	Length: 8 Type: CHA Bytes: 8	Length: 8 Type: CHA Bytes: 8
TIMESTAMP	Length: 26 Type: CHA Bytes: 26	Length: 26 Type: CHA Bytes: 26	Length: 26 Type: CHA Bytes: 26	Length: 26 Type: CHA Bytes: 26
NUMERIC	Length: 18 Type: PACK Bytes: 1-10	N/A	N/A	N/A

Note: To find how data is represented in VisualAge Generator for each database manager and operating system you are using, locate the column with the database manager you are using and the row with the appropriate SQL Data Type. The VisualAge Generator representation is displayed in the table cell that is common to both.

1. C++ does not have a packed decimal data type. Floating point variables with a maximum precision of 15 digits are used to access decimal columns. SQL columns of up to 18 digits can be accessed, but the numbers will be accurate only to the first 15 significant digits.

Note: For workstation systems, GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC are supported only on DBCS-enabled systems.

To view data conversions for Oracle, see Table 15 on page 76.

If any data type not shown on the table is returned from the SQL database manager, the column is defined as type HEX with a byte length corresponding to the byte length returned from the database manager. The SQL data type is saved in the record definition.

The SQL DECIMAL data type has both precision (number in digits) and scale (number of decimal places to the right of the decimal) defined. When defining a data item in VisualAge Generator the length and the decimal places will map to precision and scale for SQL decimal data types. Therefore, if you know the precision and scale in SQL, you know the length and decimal places in VisualAge Generator, and vice versa.

The SQL types VARCHAR and VARGRAPHIC are variable-length fields, but a maximum length is specified when they are defined. The maximum length is used for the VisualAge Generator data item byte length. LONG VARCHAR and LONG VARGRAPHIC columns are not defined with a maximum VisualAge Generator size in the SQL table definition; therefore, the SQL type maximum length is used for the data item byte length in VisualAge Generator.

These large fields can result in excessive virtual storage requirements for the program. A warning is issued about the size of these fields when the definition is being converted to an SQL row record definition. You should change the byte-length to the maximum size that the column actually contains.

VisualAge Generator-to-SQL data type conversion

When the test facility accesses columns in a relational database, it sets the SQL data type in the SQL descriptor area (SQLDA) according to the way the item is defined in the SQL row record. The SQL type used for each VisualAge Generator data type is listed in Table 10.

Note: When setting the SQL data type, VisualAge Generator uses DB2 specific data types. When non-DB2 database management systems (DBMS) are used during test, the DB2 data types are mapped to the data types supported by the accessed DBMS.

The VisualAge Generator data types in the table are the only data types supported in SQL row records.

Table 10. Valid Data Types for Use in SQL Row Records

VisualAge Generator type	Bytes	SQL type	Code
CHA	1–32767	User specified	
DBCS	2–32767	User specified	
BIN	2	SMALLINT	501
BIN	4	INTEGER	497
PACK[1]	10	DECIMAL	485
HEX	1–32767	User specified	

Note: C++ does not have a packed decimal data type. Floating point variables with a maximum precision of 15 digits are used to access decimal columns. SQL columns of up to 18 digits can be accessed, but the numbers will be accurate only to the first 15 significant digits.

VisualAge Generator generates COBOL or C++ host variable definitions that are appropriate for the SQL code, according to the way the item is defined in the SQL row record.

Neither VisualAge Generator SQL row records nor the database managers support the MIX data type. However, they support mixed data in SQL character (CHAR, VARCHAR) columns. The program can move mixed data between MIX fields on maps and CHA items in SQL row records. DB2/2 supports mixed data only on DBCS-enabled systems.

For more information on mixed data in SQL programs, see "Chapter 9. Developing programs containing DBCS" on page 213.

Support for FLOAT and other unknown data types

VisualAge Generator supports FLOAT and any other SQL data types that do not have a corresponding VisualAge Generator data type by using HEX data items. When VisualAge Generator extracts a column definition with an unrecognized data type from the SQL catalog, the item type for that column is set to HEX. The SQL data type and column length are saved with the record definition. The program can reference that item in MOVE and LOGICAL COMPARE statements but not in calculations.

If you do not have access to the SQL database during program definition, you can set the SQL data codes for HEX items on the SQL Row Item Usage window. If you wanted to define a FLOAT column in an SQL row, you define the item as a HEX item with length 8 and set the SQL data code to 481. The SQL data codes for SQL data types are listed in the SQL program manuals in the description of SQLDA.

Columns with the following SQL data types cannot be accessed in a generated program because there is not an equivalent COBOL data type for the item:

- 460/461 - Null terminated character string
- 476/477 - Varying-length character string (Pascal)

Decimal columns must not be accessed as HEX items, use PACK items for decimal columns.

Using DATE, TIME, and TIMESTMP SQL Columns

DATE, TIME, and TIMESTMP columns in the database are associated with CHA data items in record and map definitions. The values are in character format with separator characters when processed in the program.

To have the map edit routines and the EZEDTELC special function word produce dates in the format expected by the database manager, set the VisualAge Generator installation default Gregorian date format for long dates (four digit year) equal to the date format specified for the database manager.

Define date items as 10 character fields. Then when the program uses the date map item edit to set the value of the field or moves EZEDTELC to the field, the resulting date is in the correct format for storing in the database.

The method of setting the default date format varies with the runtime environment as shown in Table 11.

Table 11. Methods for Setting the Default Date Format

Environment	How default date format is set
MVS, VSE, VM	Specified during IBM VisualAge Generator Server for MVS, VSE, and VM Installation. See the installation program directory for more information.
OS/400	Derived from date format and date separator job options at runtime.
OS/2, AIX, CICS for AIX, Windows NT, CICS for Windows NT, HP-UX, Solaris, and CICS for Solaris	Specified in the EZERGRGL_xxx environment variable

SQL Nulls

SQL supports null values for columns in an SQL table row. A null value indicates the absence of a value for the column; that is, the contents of the column do not exist. A null value can occur for any of the following reasons:

- A row is added to a table without specifying all of the columns
- A new column is added to the definition of an existing table
- An update explicitly sets the column to a null value

VisualAge Generator supports the following statements that can be used with data items in SQL row records:

- TEST item NULL
- IF item IS NULL
- IF item NOT NULL
- WHILE item IS NULL
- WHILE item NOT NULL
- SET item NULL

Items in tables and records other than SQL rows cannot have a null value. SET item NULL is supported only for data items in SQL row records.

Columns with a null value that are read from the SQL database have item contents initialized in storage according to their data type (blanks for character and DBCS data, zeros for numeric and hexadecimal data). Movement from any non-SQL item or any non-null SQL item into an SQL row item makes the SQL item not null.

Movement from a null SQL item to an SQL row item makes the target item null and sets it to its initial value (blanks or zeros). Moving a null SQL item to a non-SQL item sets the target item to blanks or zeros. The null value indicator is not carried along on the move. Moving the same data back to the SQL item makes the SQL item blank or zeros, and not null.

Comparisons between record items do not consider null values. For comparisons, items with null values are set to binary zeros. Use the IF and TEST statements to determine if an item has a null value.

The null indicators that are used with SQL are included as part of every SQL row record item structure; they consist of 2 bytes before each SQL data item. That is, every SQL item passes 2 bytes that are not null during SQL to VisualAge Generator conversion. You are not aware of these indicators unless an SQL row is passed as a parameter or redefined.

Note: Null indicators are not included in the generated WHERE clause.

The details of the record item structure used for SQL data are in "Storage layout of SQL row records".

Storage layout of SQL row records

You do not need to be aware of the internal layout of an SQL row record except when using the following functions:

- Receiving an SQL row as a parameter in a non-VisualAge Generator program
- Receiving an SQL row as a parameter in a VisualAge Generator program when the received record is defined as something other than an SQL row
- Redefining an SQL row record

If you use any of these functions, you need to know the following internal layout of the record structure for an SQL row in storage. Figure 2 on page 36 illustrates the internal layout of an SQL record.

- All record data structures start on a word boundary. A further boundary alignment within a record structure does not exist.
- The data items appear in the buffer in the order they were defined.
- Each data item in an SQL row record is preceded by 4 bytes of data.
  - The 2 leading bytes are the null indicator. A null indicator is reserved for all fields, even when the table column associated with the field does not permit nulls. Any negative binary value (bit 0 is on) is a null value. Any other value is not a null value.
  - The second pair of bytes is used as a length field for some items when running in static mode. The contents of this area are unpredictable.

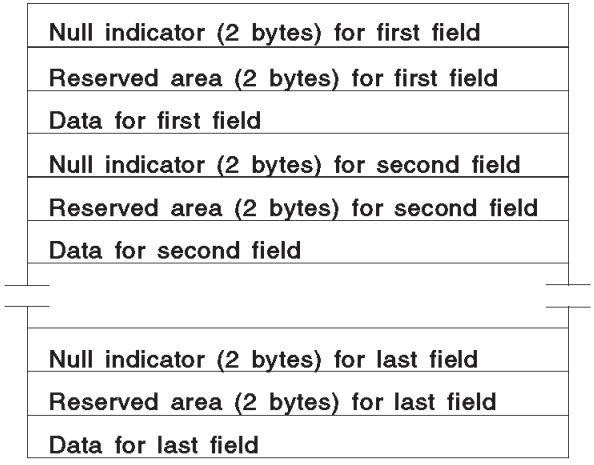

Figure 2. The internal layout of an SQL record

If you change an SQL row record to a different organization, filler fields are optionally added to the structure so that the structure matches the layout of the SQL row record as it appeared in internal storage. The LEVEL for each item is set to 3, and the OCCURS value for each item is set to 1.

If you change a record with different organization to an SQL row record, the following items are deleted from the record's data item structure:

- All filler items
- All items that are substructured. For example, if a level 3 item contains level 4 items, the level 3 item is deleted and the level 4 items are kept. If another level 3 item has no substructure, that level 3 item is not deleted.

The LEVEL for each remaining item is set to 3, and the OCCURS value for each item is set to 1.

If you want to test or set the null indicator items in the internal structure directly, you can define a new record that is a record redefinition of the SQL row record. The redefined record structure must match the internal record structure of the SQL row. You can create an exact redefinition of the SQL row internal structure by copying the SQL row Repository/ENVY library part into a new record part and then changing the record specification for the new part to be a redefined record for the original SQL row. Data definition adds filler fields to the redefined record structure that match the null indicator and length fields in the SQL record buffer.

Defining SQL programs

The first step in defining an SQL program is specifying the SQL row definitions. This is usually done by a database administrator. If you are a program developer, you can access the SQL row definitions by loading the database administrator's ENVY package/application. You can also define the SQL row records in your own ENVY package/application, if that is the practice at your installation.

The definition of maps and working storage is the same for an SQL program as for any other VisualAge Generator program. Defining the program logic is also very similar to the definition of program logic for a file-based program.

During program logic definition, you define the set of functions to run for the program. Defining the I/O functions for an SQL program is similar to defining the I/O functions for a file-based program. You use I/O options and specify SQL rows as I/O objects. VisualAge Generator creates the SQL statement necessary to run a function with an SQL row object. For certain I/O options, you can modify the SQL statement to request additional SQL functions.

Relationship between functions and SQL statements

VisualAge Generator generates SQL statements for running all I/O options, except SQLEXEC. Table 12 lists the SQL statements used for each I/O option.

You can use the SQL Statement Definition window in a function definition to view the primary SQL statement that VisualAge Generator uses for the function. You can modify the following SQL statements:

- SQL SELECT statement, except the FROM clause, built for INQUIRY, UPDATE, SETINQ, or SETUPD functions
- SQL INSERT statement, except the table name, built for an ADD function
- SET clause of the SQL UPDATE statement built for a REPLACE function

The SQLEXEC function enables you enter your own SQL statements using the Object Selection function. Table 12 illustrates the relationship between the I/O option and the SQL statement used to run the I/O option.

Table 12. I/O Option and SQL Statement Relationships

I/O Option	SQL Statements used to Run the I/O Option
INQUIRY (Single-row select=yes)	SELECT
INQUIRY (Single-row select=no)	SELECT through DECLARE CURSOR OPEN FETCH CLOSE

Table 12. I/O Option and SQL Statement Relationships (continued)

I/O Option	SQL Statements used to Run the I/O Option
UPDATE	SELECT through DECLARE CURSOR OPEN FETCH CLOSE (only if SQLCODE=100, no more rows)
SETINQ	SELECT through DECLARE CURSOR OPEN
SETUPD	SELECT through DECLARE CURSOR OPEN If you specified the cursor to declare with hold: SELECT through DECLARE CURSOR WITH HOLD
SCAN	FETCH CLOSE (only if SQLCODE=100, no more rows)
ADD	INSERT
REPLACE	UPDATE WHERE CURRENT OF cursor CLOSE (if REPLACE follows UPDATE function)
DELETE	DELETE WHERE CURRENT OF cursor CLOSE (if DELETE follows UPDATE function)
CLOSE	CLOSE
SQLEXEC	User-defined statements

Accessing relational tables without coding SQL statements

This section describes how to perform basic SQL functions without coding SQL statements by using I/O options and letting VisualAge Generator generate default SQL statements.

Accessing single rows by key items

If the SQL row record has been defined with key items that represent columns in the table with a unique index, then you can access specific rows directly by keys just as if the record were in an indexed file.

Reading a specific row: To read a specific row, use an INQUIRY function and specify the SQL row record name as the I/O object. Move the key values for the row you want to read into the key items before the I/O option is invoked.

Changing a specific row: To change a specific row, move the key values for the desired row into the key items and use an UPDATE instead of an INQUIRY function to read the row. After the row is read, change any fields in

the row except the keys and write the row back to the database using a REPLACE function with the SQL row record as the I/O object.

Adding a specific row: To add a specific row to the database, move data into the record items, including the key values into the key items, and use an ADD function to insert the row in the database.

Deleting a specific row: To delete a row, move the key values for the row to be deleted into the key items, read the row with an UPDATE function, and follow the UPDATE with a DELETE function using the SQL row record as the I/O object.

Releasing a specific row: If your program uses UPDATE to read a row and then determines that the row should not be replaced or deleted, use a CLOSE function to release the row that was obtained for update.

If a CLOSE is not issued, the row is held either until the next I/O option for the same record, or the next commit point, or the end of the main program.

Processing selected sets of rows using SETINQ, SETUPD, and SCAN

To process a selected set of rows from a table, use a SETINQ or SETUPD function to select the rows. Then use a SCAN function within a loop to retrieve the rows one at a time. The SCAN can be followed by a REPLACE or DELETE function if the rows are selected using SETUPD.

Using SETINQ or SETUPD to select the rows: To use the SCAN I/O option with an SQL row record, you must first select a set of rows for retrieval by using the SETINQ or SETUPD I/O option. SETINQ and SETUPD do not retrieve any data, but only identify a set of rows for later retrieval using a SCAN function loop. The rows selected depend on whether you specified key items or selection conditions for the rows. Before the SETINQ or SETUPD option runs, your program should move the required key values to the key items or to any host variables used in the selection conditions.

Note: If you specify multiple data items as keys for a record, the default WHERE clause consists only of the default selection conditions for the record.

Table 13 shows the rows selected by the default SELECT statement built for SETINQ or SETUPD functions.

Table 13. Rows Selected by Using Default SETINQ or SETUPD Function

	No Selection Conditions	**Selection Conditions**
No Key Item Specified or Multiple Key Items Specified	Select all rows in the table	Select all rows that meet selection conditions

Table 13. Rows Selected by Using Default SETINQ or SETUPD Function (continued)

	No Selection Conditions	Selection Conditions
Single Key Item Specified	Select all rows with a key greater than or equal to the key item	Select all rows with a key greater than or equal to the key item and that meet selection conditions

The SETINQ function sorts the selected rows in key sequence order if key items were specified. The order of the key items in the SQL row record determines the sequence of the keys in the ORDER BY clause. The SETUPD function does not sort rows, due to an SQL restriction, but does select rows for update, meaning that a REPLACE or DELETE can optionally follow the SCAN function that actually reads the rows.

To change the rows selected, you can modify the SELECT statement built for SETINQ or SETUPD during function definition.

Using SCAN to read one row at a time: A SCAN function retrieves the next row in a selected set of rows. The columns read by the SCAN are the columns specified in the SELECT statement for the previously run SETINQ or SETUPD function. The columns are read into the host variables identified in the INTO clause of the SELECT statement.

To process all rows in the selected set one at a time, put the SCAN I/O option inside a WHILE loop. If the SCAN function is run, when no rows are left in the set, the NRF (no record found) indicator is set for the record.

Figure 3 on page 41 shows the logic used to scan through the Inventory table and print a line on a report for each inventory item in the table. The logic is defined as a function with invoked I/O functions. All the processing statements are defined in the group to make the logic easier to understand.

```
MOVE 0 TO FINISH;         /* Set stop switch
MOVE 0 TO INVTABLE.PARTNO;/* Set key item to low value to select all rows
                          /* Use SETINQ function to select all rows with key >= 0
PSETINQ();                /* Option SETINQ, object INVTABLE
                          /* Use SCAN function to read each row in the table
WHILE FINISH EQ 0;        /* Loop until done
 PSCAN();                 /* Option SCAN, object INVTABLE
 IF INVTABLE IS NRF;      /* Check for last record found
  MOVE 1 TO FINISH;       /* end loop
 ELSE;                    /* If we found a valid record
  PPRINT();               /* Go print the contents of the row
 END;
END;
```

Figure 3. Sample Logic: Reading Each Row in the Inventory Table

Releasing selected rows using CLOSE: The set of rows selected with a SETINQ or SETUPD I/O option must be released when the program has finished processing the set. If your program loops through the entire set until the no record found (NRF) condition is encountered, VisualAge Generator automatically releases the set for you by issuing an SQL CLOSE statement.

If your program ends the SCAN loop before reaching the end of the set of rows, you must use a CLOSE statement to release the rest of the set. If a CLOSE statement is not issued, the rest of the set is held until the next INQUIRY, UPDATE, SETINQ, or SETUPD I/O option for the same object, or until the next commit, or until the main program ends.

Figure 4 on page 42 demonstrates when a CLOSE I/O option is required. The logic of the example calculates the total parts on order for a specific part in the INVQUOTS table. The logic also prints a line showing the parts-on-order number for each supplier of the part. The processing loop can end after the last row for the part has been processed because the set of rows returned by the SETINQ is sorted on the key field.

```
MOVE 0 TO FINISH;     /* Set stop switch
MOVE 0 TO TOTALQN;    /* Set initial quantity to zero
MOVE MAPIN.PARTNO TO INVQUOTS.PARTNO; /* set key item from map input
                      /* Use SETINQ function to select all rows with
                      /*  key >= key from map
PSETINQ();            /* Option SETINQ, object INVQUOTS
                      /* Use SCAN function to read each row in the table
WHILE FINISH EQ 0;    /* Loop until done
 PSCAN();             /* Option SCAN, object INVQUOTS
 IF INVQUOTS IS NRF;  /* Check for last record found
  MOVE 1 TO FINISH;   /* End loop
 ELSE;
  IF INVQUOTS.PARTNO NE MAPIN.PARTNO; /* Check for new part number
   MOVE 1 TO FINISH;                  /* end loop
                                      /* Use CLOSE function to release
                                      /*  remaining rows in set
   PCLOSE();                          /* Option CLOSE, object INVQUOTS
  ELSE;                               /* If we found a row for this part
                                      /*  number
   PPRINT();                          /* Print number of parts for this
                                      /*  supplier
   TOTALQN = TOTALQN + INVQUOTS.QONORDER; /* Increment quantity on order
  END;
 END;
END;
PPRTTOT();                  /* Print total number of parts
                            /*  for part number
```

Figure 4. Sample Logic: Retrieving All Suppliers of a Specific Part

Replacing or deleting a row retrieved by using a SCAN function: If a row was selected using a SETUPD, the SCAN function can be followed by a REPLACE function to modify the row or a DELETE function to delete the row.

Figure 5 on page 43 shows logic that scans through the Quotations sample table and changes the delivery time for a specific supplier. If a part number was specified in the map input, only the delivery time for that part is changed. The logic assumes you have defined an SQL row record named QUOTES for the table with key item SUPPNO. Note that you must scan all rows returned by the SETUPD function, because the set of rows returned is not sorted due to an SQL restriction.

```
MOVE 0 TO FINISH;                  /* Set stop switch
MOVE MAPIN.SUPPNO TO QUOTES.SUPPNO;/* Set key item from map input
                                   /* Use SETUPD function to select
                                   /*  all rows with key >= key from map
PSETUPD();                         /* Option SETUPD, object QUOTES
                                   /* Use SCAN function to read each
                                   /*  row in the table
WHILE FINISH EQ 0;                 /* Loop until done
 PSCAN();                          /* Option SCAN, object QUOTES
 IF INVQUOTS IS NRF;               /* Check for last record found
  MOVE 1 TO FINISH;                /* end loop
 ELSE;
  IF INVQUOTS.SUPPNO EQ MAPIN.SUPPNO;      /* If we found the correct
                                           /*  supplier
   IF MAPIN.PARTNO EQ 0                    /* If we are changing all
                                           /*  parts for supplier
    OR MAPIN.PARTNO EQ INVQUOTS.PARTNO;    /* Or this is the specified
                                           /*  part
    MOVE MAPIN.DELIVERY TO QUOTES.DELIVERY;/* Reset delivery time
                                           /* Use REPLACE function to
                                           /*  replace row in the table
    PREPLAC();                             /* Option REPLACE,
                                           /*  object QUOTES
   END;
  END;
 END;
END;
```

Figure 5. Sample Logic: Modifying Delivery Time for Parts from a Supplier

If you understand SQL statement syntax, you might simplify the logic in this example by modifying the WHERE clause in the SELECT statement for the SETUPD function to select only the rows where the supplier number was the supplier number entered on the map. "Modifying SQL statements" on page 44 contains more information on modifying the SQL statements built by VisualAge Generator.

Searching on a partial key: If you only know the first part of a row key, you can retrieve the first row that has a key beginning with that partial key by doing the following:

1. Move the partial key into the key item.
2. Use a SETINQ function to select and sort all records with keys greater than or equal to the key item value.
3. Use the SCAN function to retrieve the first row in the set.
4. Use the CLOSE function to release the rest of the rows in the set.

Alternatively, you might use an INQUIRY I/O option and modify the SQL statement built for the option to select rows with key greater than or equal to

the key item value. The INQUIRY then returns the first row of the selected set and automatically releases the rest of the rows in the set.

Updating a set of records in key sequence order: Updating a set of records in key sequence order requires special logic because SETUPD cannot sort rows due to a database manager restriction. To perform the operation, you define a second SQL row record as an alternate specification of the first. Use SETINQ and SCAN to retrieve the set of rows in key sequence order. To replace a row that has been retrieved, move the key from the original row to the key of the alternate specification record. Then, issue an UPDATE and REPLACE sequence to retrieve and update the row using the alternate specification record.

Modifying SQL statements

On the SQL Statement Definition window, you can view or modify the SQL statement that VisualAge Generator creates for the I/O option.

Modifying the SQL statement for a function

During function definition, when the I/O object is an SQL row record, you can modify most of the clauses generated by VisualAge Generator for SQL functions.

You can modify any of the lines on the SQL Statement Definition window that are not preceded by an asterisk (*). If you understand SQL syntax, you can change the row selected and the columns read from the table. Table 14 summarizes VisualAge Generator support for modifying default clauses:

Table 14. VisualAge Generator Support for Default Clause Modification

I/O Option	Clauses You Can Modify	Clauses You Cannot Modify	Clauses You Can Delete
INQUIRY SETINQ	SELECT INTO WHERE GROUP BY HAVING ORDER BY	FROM	WHERE GROUP BY HAVING ORDER BY
UPDATE SETINQ	SELECT INTO WHERE FOR UPDATE OF	FROM	WHERE
ADD	INSERT INTO VALUES	Table name	None

Table 14. VisualAge Generator Support for Default Clause Modification (continued)

I/O Option	Clauses You Can Modify	Clauses You Cannot Modify	Clauses You Can Delete
REPLACE	SET	UPDATE WHERE CURRENT	None
DELETE	None	DELETE FROM WHERE CURRENT	None
SCAN	None	FETCH	None
CLOSE	None	CLOSE	None

In changing a statement, use the SQL statement syntax described in the appropriate relational database manager reference manual with the following additions or exceptions:

- You can use data items as host variables in the statement by coding a colon immediately preceding the data item name. The data item name can be qualified and subscripted. The subscript can be a numeric data item or a numeric literal. At run time, when the SQL statement is processed by the database manager, the host variable in the statement is replaced by the value of the variable.

  Any data item in the program whose length and type are compatible with SQL can be used as a host variable. HEX items used as host variables must be from an SQL row record.

  The colon (:) is the default SQL host variable indicator, that is :item-name format. You can change the host variable indicator on the **SQL** options/preferences.
- Do not define host variables for null indicator checking. Null indicators are maintained by VisualAge Generator for all items in SQL row definitions (see "SQL Nulls" on page 34 for more information). You can use the TEST and IF statements to test and the SET statement to set null indicators for SQL row items. Null indicators are not maintained for items outside of SQL row definitions.
- An INTO clause is shown with all SELECT commands. The SELECT command is actually run with an SQL cursor. The INTO clause identifies the host variables that receive the data when a row is retrieved with the FETCH command associated with the cursor. The INTO clause is defined with the SELECT command because a one-to-one relationship must be maintained between the selected columns and the items in the INTO clause. You can avoid the use of the cursor for INQUIRY I/O options by specifying YES for the Single Row SELECT option.

Note: Single row SELECT is not supported when table name host variables are used in the statement or if the **Execution Time Statement Build** check box is selected for the function.

- You can define a comment line in the statement by putting a semicolon (;) as the first character in the line or by coding a /* as the first 2 characters in the line.
- You can enter an SQL column name in host column name form by coding an SQL row item name for the SQL column immediately preceded by an exclamation point (!). The item name cannot be qualified or subscripted. The item must be one of the items defined in the record definition for the I/O object.

  When the SQL statement is prepared for run time, the data item name is replaced by the SQL column name defined for the data item in the SQL row definition.

  You can change the SQL column name indicator on the **SQL** options/preferences.
- When the WHERE clause of an SQL SELECT statement is modified to include the LIKE or IN predicates, VisualAge Generator Developer and VisualAge Generator Server for MVS, VSE, and VM strip trailing blanks from character and DBCS fields used as host variables. This can affect the results of the search performed by SQL.

Checking the syntax of a modified SQL statement

If you modify an SQL statement, VisualAge Generator validates the host variable names entered in :item-name format and the column names in !item-name format. VisualAge Generator then checks that the INTO clause of a SELECT statement consists of a list of host variables separated by commas, but leaves all further statement syntax checking to the database manager.

On the SQL Statement Editor window, select **Validate** to request an immediate syntax check of the statement by the database manager using the SQL dynamic PREPARE function. Before the prepare the following is true:

- Each host variable is replaced in the statement with a question mark (?) or parameter marker
- Each column name entered in !item-name format is replaced with the actual SQL column name
- The INTO clause is deleted from SELECT statements

If the SQL database manager finds an error in the statement, it returns an error code and error information in the SQL Request Error window. Some database manager functions are not valid with an SQL dynamic prepare function. Use of these functions results in error conditions when the database checks SQL syntax.

Resetting a modified SQL statement to the default SQL statement

If you want to reset a modified SQL statement to its default definition, display the SQL statement and from the **Options** pull-down menu, select **Reset SQL Statement**.

If you want to reset the statement to what it was before editing, on the SQL Statement Definition window, select the **Cancel** push button.

Effects of SQL row record changes on modified SQL statements

Changes to an SQL row record definition are automatically reflected in default SQL statements built for a function. Any modifiable clauses in a modified SQL statement are not built again if the SQL row definition for the I/O object is modified. You must manually update modified statements to reflect the new record definition if changes are required.

Defining SQL statements using the SQLEXEC I/O option

The SQLEXEC I/O option is used for advanced SQL programming functions for database manipulation. SQLEXEC supports functions that write to the relational database. With SQLEXEC, you define the entire SQL statement. You can enter any statement that you can run using the EXECUTE statement of the SQL interface for high-level languages. These include INSERT, UPDATE, and DELETE statements in single- or multiple-row format, GRANT and REVOKE statements for authorization processing, and CREATE and DROP statements for table, view, index, and synonym definition.

The major statements that cannot be issued using the SQL EXECUTE statement are any statements that read data from the database (SELECT, SCAN, OPEN, CLOSE), and COMMIT and ROLLBACK WORK statements. For SELECT processing, use the INQUIRY, UPDATE, SETINQ, or SETUPD functions and modify the SQL statement built for the function. Use the EZECOMIT and EZEROLLB special function words for COMMIT and ROLLBACK processing.

Entering the SQL statement for the SQLEXEC I/O option

Enter the SQL statement in the free-format line edit area on the panel. Use the statement syntax described in the appropriate relational database manager reference manual.

You can define host variables as part of the SQL statement. You can also define host column names using the !item-name format if you have specified an SQL row record as the I/O object. “Modifying the SQL statement for a function” on page 44 contains a description of host variables and host column names in SQL statements. “Using table name host variables with the execution time statement build option” on page 53 contains more information on using table name host variables with SQLEXEC functions.

You can use the SQLEXEC I/O option to issue multiple-row INSERT, DELETE, and UPDATE statements that are not supported directly with VisualAge Generator I/O options. Examples of each of these statements are as follows:

Multiple-row insert

```
INSERT INTO DEPT.EMPTAB
       SELECT EMPLOYEE_NUMBER, WEEKLY_SALARY
       FROM COMPANY.EMPTAB
       WHERE DEPARTMENT_NUMBER = :EMPTAB.DEPTNO
```

Multiple-row delete

```
DELETE FROM COMPANY.EMPTAB
       WHERE EMPLOYEE_NUMBER = :EMPLNO
```

Multiple-row update

```
UPDATE COMPANY.EMPTAB
       SET WEEKLYSALARY = WEEKLYSALARY * 2
       WHERE EMPLOYEE_NUMBER = :EMPLNO
```

Using SQLEXEC to issue data definition statements

You can use SQLEXEC to issue the following SQL data definition language statements:

- GRANT
- REVOKE
- CREATE TABLE
- DROP TABLE
- CREATE INDEX
- DROP INDEX
- CREATE SYNONYM
- DROP SYNONYM
- CREATE VIEW
- DROP VIEW

If you want to use host variables in these statements, select the **Execution Time Statement Build** check box as described in "Using execution time statement build with SQLEXEC functions" on page 50.

SQL statements not supported by SQLEXEC

You cannot issue the following commands using the SQLEXEC function:

- SELECT
- INCLUDE SQLCA
- INCLUDE SQLDA
- WHENEVER
- OPEN
- CLOSE
- FETCH

- DECLARE CURSOR
- COMMIT WORK
- ROLLBACK WORK
- CONNECT
- PREPARE
- EXECUTE
- EXECUTE IMMEDIATE
- DESCRIBE

Controlling SQL statement preparation with execution time statement build

SQL statements must be preprocessed before they can be performed by the database manager. The SQL Statement Definition window used for SQLEXEC functions and SELECT statement functions contains the **Execution Time Statement Build** check box that enables you to control when and how SQL statements are presented to the database manager for statement preparation. The Execution Time Statement Build option permits an SQL statement generated for an SQL function to be dynamically modified at run time. It also permits host variables to be used in SQL where SQL does not normally support host variables.

If you do not select the **Execution Time Statement Build** check box, the SQL statement cannot be modified by the program at run time. You can use host variables as defined in normal SQL statement syntax. All valid host variable data types are supported. "Using table name host variables with the execution time statement build option" on page 53 contains special considerations for table name host variables.

If you select the **Execution Time Statement Build** check box, the statement is prepared each time the function is run. The SQLEXEC function is implemented using the EXECUTE IMMEDIATE command. INQUIRY, SETINQ, UPDATE, and SETUPD functions are implemented using PREPARE and cursor manipulation statements. The SQL statement is built each time the function is run by replacing all the host variables in the statement (except the host variables in the INTO clause of the SELECT statement) with the character representation of the contents of the host variables.

Only host variables with CHA, BIN, or PACK data types can be used. The CHA fields are inserted directly in the statement without being enclosed in single quotes. This is an advantage because it permits host variables to be used in places where SQL does not normally support host variables.

Statements that contain host variables cannot be checked for syntax errors by the test facility or used in a generated program when the **Execution Time Statement Build** check box is selected.

Selecting the **Execution Time Statement Build** check box is not valid when generating using the ANSI SQL format.

Using execution time statement build with SQLEXEC functions

When you select the **Execution Time Statement Build** check box for SQLEXEC, the function is run using the EXECUTE IMMEDIATE interface. The statement string that is run is created by replacing the host variables in the statement with the character representation of the contents of the host variables. The replacement is done before the string is passed to the database manager.

Only CHA, BIN, and PACK items can be used as host variables for the EXECUTE IMMEDIATE interface statement without being enclosed in quotation marks. This permits the EXECUTE IMMEDIATE interface to be used with SQL statements that do not normally support host variables.

For example, you can define the following GRANT statement for an SQLEXEC function and run it with the Execution Time Statement Build option:

```
GRANT :TYPE
      ON :TABLENAM
      TO :USERID
```

If your program moved 'SELECT' to the TYPE date item, 'COMPANY.EMPTAB' to TABLENAM, and 'SMITH, JONES' to USERID, and then ran the SQLEXEC function, the following SQL statement is issued using EXECUTE IMMEDIATE:

```
GRANT SELECT ON COMPANY.EMPTAB TO SMITH, JONES
```

Note: CHA host variables are not surrounded with single quotation marks when the statement string is built for run time. If you intend the CHA host variable to be used as a character constant value in the SQL statement, you are responsible for surrounding the character value with single quotation marks.

Using execution time statement build for dynamic SELECTs

You can select the **Execution Time Statement Build** check box for SELECT statement functions in addition to SQLEXEC function SQL statements. When the option is selected, the SQL statement is prepared dynamically each time the function is run. Any REPLACE or DELETE functions associated with the SELECT are also run dynamically.

For all clauses except the INTO clause in SELECT statements, the statement text to be prepared is built by replacing the host variables in the statement with the character representation of their contents. For these clauses, only host variables with type CHA, BIN, or PACK can be used. The CHA fields are

inserted directly in the statement without being enclosed in single quotation marks. This has the advantage of permitting host variables to be used in places where SQL does not normally support host variables.

Example of a dynamic WHERE clause

When Execution Time Statement Build is used, host variables can be defined in place of a WHERE clause, enabling dynamic specification of selection criteria within a program. The syntax check function cannot be used with statements defined in this way because the SQL statement to run is not known until run time.

Before the function EXAMPLE is run, the program must move a valid WHERE clause to the host variable WHRCLS. For example:

```
MOVE "PARTNO=1 and DESCRIPTION='BOLT'" TO WHRCLS ;
```

Then, the resulting SQL SELECT statement is prepared as shown in the following example.

Note: The CHA host variables are not surrounded with single quotation marks when the statement string is built for preparation. If you intend to use a CHA host variable as a character constant value in the SQL statement, the program is responsible for surrounding the character value with single quotation marks.

```
SELECT
  PARTNO, DESCRIPTION, QONHAND
INTO
  :PARTNO, :DESCRIPTION, :QONHAND
FROM
  SQLDBA.INVENTORY
WHERE
  :WHRCLS
/** INSERT ORDER BY CLAUSE HERE **
```

The example below shows how the contents of WHRCLS are built into the SQL statement. The statement is then dynamically prepared.

```
SELECT
  PARTNO, DESCRIPTION, QONHAND
FROM
  SQLDBA.INVENTORY
WHERE
  PARTNO=1 AND DESCRIPTION='BOLT'
```

Note: Nesting of host variables in dynamically-built clauses is not supported. For example, if the following statement is run before the I/O option in the example code above, an SQL error is returned from the database manager:

```
MOVE "PARTNO = :PARTNO" to WHRCLS ;
```

Example of a dynamic ORDER BY clause

You can define a host variable in place of an ORDER BY clause to enable the program to dynamically change the sort column. The sort column determines the order the selected rows are retrieved and presented to the program user.

For example, using the Execution Time Statement Build option you can define an SQL statement to retrieve all the parts in the parts table in the following order:

```
SELECT
  PARTNO, DESCRIPTION, QONHAND
INTO
  :PARTNO, :DESCRIPTION, :QONHAND
FROM
  SQLDBA.INVENTORY
ORDER BY
  :ORDERCOL
```

The program can move the column number of the sort column to the ORDERCOL item before the SETINQ option is run. For example, to sort on the description column, the program should contain the following MOVE statement, assuming ORDERCOL is a character item:

```
MOVE "2 ASC" TO ORDERCOL;
```

Then the following statement is dynamically prepared when the SETINQ I/O option is run:

```
SELECT
  PARTNO, DESCRIPTION, QONHAND
INTO
  :PARTNO, :DESCRIPTION, :QONHAND
FROM
  SQLDBA.INVENTORY
ORDER BY
  2 ASC
```

Example of changing a static SQL statement to a dynamic SQL statement

You cannot change a static SQL statement that uses CHA host variables to a dynamic SQL statement without modifying the program logic that sets the value in the CHA host variables. This limitation is because the value in the CHA variables must be enclosed in single quotation marks for correct dynamic preparation of the SQL statement.

For example, suppose the following SQL statement was working in static mode:

```
SELECT
  PARTNO, DESCRIPTION, QONHAND
INTO
  :PARTNO, :DESCRIPTION, :QONHAND
```

```
FROM
  SQLDBA.INVENTORY
WHERE
  DESCRIPTION = :DESCRIPT
```

For the program to continue to work correctly when the Execution Time Statement Build option is selected for the SQL statement, the program must put single quotation marks around the value moved to DESCRIPT as shown in the following example:

```
MOVE "'BOLT'" TO DESCRIPT;
```

Then the following SQL statement is prepared at run time:

```
SELECT
  PARTNO, DESCRIPTION, QONHAND
INTO
  :PARTNO, :DESCRIPTION, :QONHAND
FROM
  SQLDBA.INVENTORY
WHERE
  DESCRIPTION = 'BOLT'
```

If the single quotation marks were omitted, the WHERE clause is generated as WHERE DESCRIPTION = BOLT and the SQL statement preparation is not successful because the precompiler expects a column named BOLT instead of a literal value.

Using table name host variables with the execution time statement build option

If you plan to use table name host variables with the Execution Time Statement Build option, you must consider the following:

Note: These considerations apply to all I/O options, except for SQLEXEC.

- When you do not select the Execution Time Statement Build option, you can only use table name host variables in statements that are defined on the record specified as the I/O option.
- To use a table name host variable that is not defined on the I/O object in a sub-SELECT statement, select the Execution Time Statement Build option.

For the SQLEXEC I/O option, a different consideration applies. If you do not select the Execution Time Statement Build option, the only table name host variables that can be used in the SQL statement are those defined on the I/O object. If the SQLEXEC I/O option does not have a I/O object table name, host variables can only be used when the Execution Time Statement Build option is selected.

For all other cases of using table name host variables in an SQLEXEC function, Execution Time Statement Build must be selected.

Testing the results of an SQL I/O option

To test the results of any SQL I/O request, use an IF or TEST statement. The amount of testing your program does depends on whether you want the program to end or continue processing on negative SQL return codes. It is strongly recommended that VisualAge Generator programs test the SQL record for the VisualAge Generator record I/O error values rather than EZESQCOD to determine if an I/O operation completed successfully.

Ending the program on a hard SQL error code

The default response to a hard SQL error code (304, 802, or any negative code for DB2 and DataJoiner) in the communications area is to end program processing with an error message. For DB2 and DataJoiner, the only return code your program normally needs to handle is return code 100, which indicates no more rows for the INQUIRY, UPDATE, and SCAN functions. For access using ODBC or other database management systems, check the record status, after the I/O operation, for NRF instead of return code 100.

The record status for the SQL row record is the last SQLCODE returned for the record. The no record found (NRF) status is set for DB2 and DataJoiner when the SQLCODE is 100. Your program can either test the status of the SQL row record or test the contents of the special function word EZESQCOD. The EZESQCOD special function word contains the value of the SQLCODE field in the SQLCA that was returned with the last SQL I/O function.

In the following examples, the record is the sample SQL row record, INVTABLE, defined in this chapter. To test the NRF record state, define the statement:

```
IF INVTABLE IS NRF ;/* test for record not found
```

Other special function words contain other parts of the SQLCA after an SQL I/O function.

Handling a hard SQL error code

The program does not end on a hard error SQLCODE if the EZEFEC special function word is set to 1 and an error routine is specified. In this case, the error handling logic must test for the HRD (hard error) condition, as well as, the NRF condition. You can use the IF statement to check for a hard SQL code or a specific SQLCODE as in the following example:

```
IF INVTABLE IS HRD ;/* test for hard SQLCODE
```

The status for the SQL row record is also set. When using DB2 or DataJoiner, HRD status is set for all negative SQL codes, SQL code 304, or SQL code 802. Additionally, UNQ status is set for SQL return code -803 (`insert or update was not successful because of a duplicate index column value`). ERR status is set for any nonzero SQLCODE.

SQL functions and program calls or transfers

Whenever you use XFER or DXFR statements to transfer to another program, any cursors opened with the UPDATE, SETINQ, or SETUPD I/O options will automatically be closed.

If a generated program is invoked by a non-VisualAge Generator program, the program cannot do the following:

- Issue a SETUPD or SETINQ in the called program, return to the calling program, and call the program at a later time to do the SCAN function
- Issue an UPDATE or SCAN for update in the called program and issue the REPLACE or DELETE on a subsequent call to the program

SQL locking

The database manager locks data accessed by a program until the end of a unit of work (commit or rollback) to prevent simultaneous updates or access to uncommitted changes.

The SQL database manager supports two types of locks: exclusive locks and share locks. All LUWs exclusively lock all data that they modify, and share locked data that they read. Exclusive locks prevent other users from reading or modifying the same data. Share locks permit other users to read, but not modify, the data. Generally, locks are held to the end of the LUW.

The SQL database manager automatically detects and corrects potential deadlocks. A deadlock occurs when two LUWs access data that is locked. The database manager detects these situations and 'backs out' the newest LUW. A 'back out' means that the database manager restores all changes made to the database during that LUW, and then releases the lock. The other program can then proceed. If your LUW is backed out, the database manager returns a negative SQLCODE.

Logical unit of work considerations

If a generated SQL program calls an external program developed with a tool other than VisualAge Generator, and both the program and the external program access SQL tables, the external program must not use any function that causes a commit or roll back, unless all cursors within the calling program are closed.

The following functions must not be used in the external program:

- CICS SYNCPOINT or SYNCPOINT ROLLBACK
- DL/I CHKP, ROLB, or ROLL
- SQL COMMIT WORK or ROLLBACK WORK
- OS/400 COMMIT or ROLLBACK

Programs running in non-CICS environments will commit DB2 updates only if a VisualAge Generator program has accessed an SQL table within the unit of work. In these environments, if a called external program access SQL tables, but the calling program does not, then the external program must issue an SQL COMMIT or ROLLBACK work to commit or roll back database changes.

Automatic rollback for relational database I/O

Certain error conditions returned from the database manager result in the database performing an automatic rollback for the current logical unit of work (LUW). An example of this is the deadlock condition. This rollback results in all of the active cursors in the LUW becoming non-active (or closed).

VisualAge Generator detects these specific errors and also detects that the cursors in the current LUW have been closed. Any subsequent attempt by the program to reference any affected cursor, results in the program ending. For example, if a SETUPD I/O option is followed by a SCAN I/O option, and the SCAN raises one of these error conditions, the SETUPD is not active. This is because the cursor corresponding to the SETUPD has been closed by the automatic rollback. An additional SCAN results in an error, indicating there is not an active SETUPD or SETINQ for the SCAN.

One way to avoid this program end is to check for the general error condition (ERR) after any database I/O option. If the error condition (ERR) exists then do not issue any more SCAN, REPLACE, or DELETE I/O options for the corresponding SETINQ or SETUPD.

You might also wish to define your program to check specifically for the conditions that cause an automatic rollback using the special function word EZESQCOD. These error conditions are listed in the appropriate database manager messages and codes manuals.

If your program has an active SETUPD or SETINQ I/O option, and a function against the SETUPD or SETINQ (such as SCAN, REPLACE, or DELETE) results in an error, the program can determine if the error has caused the SETINQ or SETUPD to be deactivated by examining EZESQCOD. If the value of the EZESQCOD special function word indicates that the SQL return code is one of the return codes that results in an automatic rollback, the program should not attempt another function (for example, SCAN) against the SETINQ or SETUPD. If the SQL return code does not result in a rollback, the program can continue processing as if the SETINQ and SETUPD were still active.

Assuring data integrity across transactions

If your program accesses a database that is shared by multiple users, you should define your program to run in segmented mode, or you should set the EZECNVCM special function word to 1 in your program. In both cases, the program commits database and recoverable file changes at each CONVERSE I/O option. This ensures that your program does not lock out database records for long periods of time while waiting for a response from a program user. However, you must define functions to prevent two users from changing the same record at the same time, or to reestablish the selected set of rows if your program is scanning through a set of rows.

You should use a compare and update technique to ensure that another program has not modified a row while it was being displayed to the user if the following is true:

- The LUW ends at each CONVERSE.
- The program reads a row, displays the row contents, reads changes from the map and then writes the changes back to the database.

Use the following steps to compare and update the row:

1. Read the row.
2. Save a copy of the row in working storage.
3. Display the row contents to the user.
4. Read updates from the terminal.
5. Read the row with an update function.
6. Compare the row just read with the saved row.
7. If not equal, go back to step 2.
8. If equal, update the row from the terminal input.
9. Write the row back to the database with a replace function.
10. Display a message confirming that the row has been updated.

If the row is always updated from the same system or from systems with synchronized clocks, you can include a column in the row that is set to the time of the last update. You can then simplify the preceding algorithm to save and compare only the time stamp instead of saving and comparing the entire row.

Referential integrity considerations

Databases with referential integrity affect the design of SQL programs. For example, when accessing tables which have referential constraints in database management systems which support referential integrity, your program must not violate the constraints.

IBM relational database managers have restrictions on the use of cursor-controlled UPDATE and DELETE when using referential integrity. These restrictions are documented in the appropriate database management system and database administration manuals. VisualAge Generator uses

cursor-controlled SQL UPDATE and DELETE to implement its REPLACE and DELETE I/O options. See “Relationship between functions and SQL statements” on page 37 for more information on the SQL Statements that are run for the VisualAge Generator SQL I/O options.

The following referential integrity restrictions should be considered when using the VisualAge Generator UPDATE, REPLACE and DELETE I/O options:

- You cannot use a cursor-controlled update on any of the columns of the primary key or on columns of a view derived from the primary key. You cannot use WHERE CURRENT OF with the SQL UPDATE statement, which VisualAge Generator creates for the REPLACE I/O option.
- A non-cursor update of primary key columns must not apply to more than one row.
- An SQL self-referencing table cannot be the object of a cursor-controlled delete operation. For example, a table using DELETE WHERE CURRENT OF, which VisualAge Generator creates for the DELETE I/O option, cannot be the object.

The following list includes alternative methods of performing the VisualAge Generator UPDATE, REPLACE, and DELETE I/O options within the preceding restrictions.

- You can indicate that fields in an SQL Row Record are keys. If you set the primary key fields to KEY = YES, the corresponding SQL columns are not updated. The keys are excluded from the SQL statement.
- You can update a primary key column by using the UPDATE I/O option to select the row and then define an SQL UPDATE statement using the SQLEXEC I/O option. The UPDATE in SQLEXEC is done without using a cursor.

  You can also use the SQL UPDATE statement in SQLEXEC to update the row without selecting it first. In this case, ensure that you update only one row at a time. In either case, VisualAge Generator optionally creates a model SQL UPDATE statement for you in SQLEXEC.
- To delete data from an SQL self-referencing table, you can define an SQL DELETE statement using the SQLEXEC I/O option. In this case, the delete is done without a cursor. VisualAge Generator optionally creates a model SQL DELETE statement in SQLEXEC.

Another consideration is that the database managers have error return codes for referential integrity errors. These return codes can be checked using program logic in the EZESQCOD special function word. Also, the EZESQRRM special function word might contain information on the relational table exposed and the rule or constraint in error.

If you are running programs in more than one production environment, be aware that return codes received from the different database managers can be different for the same referential integrity error. For more information about the return codes, refer to the appropriate messages and codes manual for the database manager system you are using.

Accessing distributed databases

This section describes the Remote or Distributed Unit of Work (DUW), guidelines for using EZECONCT, and default database connections. Distributed unit of work is supported only for DB2 and DataJoiner.

Remote or distributed unit of work

DB2 remote unit of work support allows a program to access one database program server per unit of work. DB2 distributed unit of work support allows a program to connect to multiple database program servers within a single unit of work. Only one connection is active at any one time (an SQL I/O option operates on the last database to which EZECONCT was issued), but database updates do not have to be committed prior to connecting to another database.

VisualAge Generator supports remote unit of work connections in all DB2 environments except CICS for VSE/ESA. Distributed unit of work connections are supported from the test facility and generated C++ programs to any DB2 database on any DB2 system.

Use the EZECONCT special function word to control the database unit of work in VisualAge Generator programs. EZECONCT lets a program connect, disconnect, or activate database connections. The parameters that can be specified on the EZECONCT call are:

```
CALL EZECONCT userid,pswd,servername,product,release,uow;
```

userid DB2 user identifier (8-byte CHA data item)

See "Authorization considerations" on page 65, for more information on options for specifying DB2 userid and password for workstation programs.

pswd DB2 password (8-byte CHA data item)

Supported on OS/2 systems with DB2/2 Version 2.1 or later, and on AIX, Windows NT, HP-UX, Solaris, VM, and VSE batch systems, but ignored on other systems.

servername
Database program server name (18-byte CHA data item).

product

Database product name (8-byte CHA item). The name of the currently connected database product is returned in this field if `servername` is blank.

The parameter is optional, but must be specified if release is specified.

release

Database product release level (8-byte CHA item). The release level of the currently connected database product is returned in this field if `servername` is blank.

The parameter is optional, but must be specified if `uow` is specified.

uow Unit of work connection option (8-byte CHA item):

R Type 1 Connect, Remote Unit of Work (default)

Perform a type 1 connection to the database identified in the `servername` parameter. Only one database can be connected at a time; EZECOMIT or EZEROLLB must be issued prior to connection to another database. Connection to another database ends an existing connection. All cursors are closed when the connection occurs.

If `servername` is RESET, a CONNECT RESET is performed. This results in a commit operation and a disconnect from the current server.

Remote unit of work must be used if the database managers are at the following levels:

- DB2 Version 2
- DB2/6000 Version 1
- DataJoiner Version 1
- Oracle

Use remote unit of work whenever your program design permits. Remote unit of work is more efficient than distributed unit of work connections.

Dxy Type 2 Connect, Distributed Unit of Work,

Perform a type 2 connection to the database identified in the `servername` parameter; `x` and `y` specify connection syncpoint and automatic disconnect options.

With type 2 connections, multiple connections can be made within a single unit of work. Connection to another database does not end prior connections. Cursors are not closed when another connection occurs.

Values for `x`, the syncpoint option, are:

1 One phase commit; only one database can be updated within the unit of work. Use one phase commit if your program design permits; a one phase commit does not have the overhead associated with a two phase commit.

2 Two phase commit; multiple databases can be updated within the unit of work.

Values for `y`, the automatic disconnect option, are:

E Disconnect must be explicitly requested. The connection remains active following a commit or rollback. A program must explicitly issue a disconnect request for connection resources to be released.

C Automatic disconnect is conditional. Connections that have no open WITH HOLD cursors are disconnected at commit or rollback.

A Disconnect is automatic. The connection is disconnected following a commit or rollback.

Specifying RESET for the `servername` is equivalent to an explicit connect to the DB2 default database named in environment variable DB2DBDFT. If the default database is not available, the connection state remains unchanged.

DISC Disconnect from the database identified in `servername`.

DCURRENT
Disconnect from the currently connected database.

DALL Disconnect from all currently connected databases.

SET Set connection to dormant database connection.

Guidelines for using EZECONCT

Follow these coding guidelines to avoid SQL errors when using EZECONCT:

- Ensure all open cursors have been closed prior to connecting to another database.
- CALL EZECOMIT or EZEROLLB prior to explicitly disconnecting from a database.
- Use SET instead of one of the type-2 connect options to reactivate a dormant database connection.

Follow these guidelines to avoid SQL errors when running a test environment with some SQL requests issued from the test facility and some from generated native C++ DLLs called locally from the test facility:

- Issue all EZECONCT, EZEROLLB, and EZECOMIT requests from a program running under the test facility, not from the generated C++ program.
- Set the default database name in the test facility profile or the DB2 environment variable DB2DFTDB. Do not set the VisualAge Generator environment variables EZERSQLDB or FCWDBNAME_<applname>.

Default database connections

If EZECONCT is not used, the default database connection is a Type 1 (remote unit of work) connection, except in the C++ runtime environments, where the default is a Type 2 (distributed unit of work) connection. The specification of the default server name varies with the environment.

Test facility uses the server name specified in the **SQL** options/preferences.

On CICS OS/2 the server name is checked for in this order:
- ELARTRDB_tttt where `tttt` is the CICS transaction identifier.
- EZERSQLDB environment variable
- DB2DFTDB

In other workstation environments the server name is checked for in this order:
- FCWDBNAME_applname where `applname` is the name of the program issuing the first SQL request
- EZERSQLDB environment variable
- DB2DFTDB (native C++ programs) or default database defined to the CICS region (recommended for best performance for CICS programs).

In MVS CICS, VSE CICS, and IMS environments, the default database is specified when setting up the environment.

For TSO, MVS batch, VSE batch, and VM programs, the default database is specified in the JCL or EXEC used to start the job. Refer to the appropriate Running manual for details.

For OS/400, the default database is the DB2/400 database on that OS/400 system.

Preparing SQL statements for the runtime environment

The SQL statements in a program must be analyzed and prepared by the database manager before they can be run. SQL statements in VisualAge Generator programs are processed in either dynamic or static mode, depending on the type of function being performed and whether the program is running under the test facility or as a COBOL or C++ program.

Dynamic Mode

Dynamic SQL statements are prepared each time the program runs. Dynamic mode is always used in the following situations:

- For all statements, when programs are running under the test facility
- When the Execution Time Statement Build option is specified for SQL statements in a generated program
- When the table name of an SQL row record is defined as a host variable
- For all statements in a generated ODBC program

In dynamic mode, the end-user or tester running the program must be authorized to use the tables referenced by the SQL statements.

Static Mode

Static SQL statements are embedded in EXEC SQL statements in the generated COBOL or C++ source and are prepared using the database precompiler (DB2, DataJoiner, or Oracle) before or during COBOL and C++ compilation. When using DB2 or DataJoiner, the output of the precompiler must be bound into a plan or a package in the SQL database. The developer performing the bind or SQL preprocessing must be authorized to use the tables referenced by the static SQL statements. The user must be authorized to use the plan or package associated with the program.

Compiling and binding the program

Programs containing SQL statements must first be precompiled. An additional step is required on some systems to bind the relational database to the program. Precompiling translates the SQL statements into COBOL or C++ statements that call the relational database manager. Binding involves putting information about the program and the SQL statements in the program into the database.

The process (and the terminology for the process) for precompiling and binding varies from environment to environment. In some environments precompiling and binding take place within a single step; in other environments precompiling and binding are two separate steps with the precompiler producing an intermediate file that is input to the bind process.

In all environments, if you move a DB2 program runtime module to another system, you must also move the program object in the database to the other system. This can be done using database manager utilities to export and import the program information; or it can be done by rerunning the bind process on the new system.

A short description of the DB2 precompile/bind process for each environment follows; however, for more detailed information on how to perform the

precompile and bind for the environment you are using, refer to the running manual for the environment you are using.

On MVS systems, precompiling and binding are separate steps. The intermediate file is called a Database Request Module (DBRM) that must be bound into an object called a plan in the DB2 database. On MVS, DBRMS from all the programs that run together as a single CICS or IMS transaction or batch job step, or that are invoked under a single DSN RUN command in MVS/TSO, must be bound into a single plan. Refer to *VisualAge Generator Generation Guide* document for information on how to set up bind commands for program preparation. See "Accessing multiple DB2 plans in CICS for MVS/ESA" on page 133 and "Accessing multiple DB2 plans in IMS" on page 135 for strategies on minimizing plan sizes for transactions.

On VSE systems, DB2/VSE preprocessing that combines precompiling and binding into one step is performed as part of preparation. The DB2/VSE preprocessor converts SQL statements in the program into an SQL package that is stored in the DB2/VSE database. The program and its associated SQL package must have consistency tokens that match. You must ensure that your program and its SQL package are created in the same preparation job stream.

On VM systems, SQL/DS preprocessing that combines precompiling and binding into one step is performed as part of preparation. The SQL/DS preprocessor converts SQL statements in the program into an SQL package that is stored in the SQL/DS database. The program and its associated SQL package must have consistency tokens that match. At run time, consistency tokens in the program and the SQL package must match before the database allows the SQL package to be executed.

On AS/400 systems, the DB2/400 processing that combines the SQL precompiling, and the ILE COBOL/400 compiling and binding into one step is performed as part of the preparation. On the AS/400, when the CRTSQLCBLI command is issued, the SQL precompiler produces a temporary source member containing information about each precompiled SQL statement. The *GEN OPTION is used on the CRTSQLCBLI command to ensure that the SQL precompiler calls the ILE COBOL/400 compiler.

If the program is a distributed program, and the relational database parameter (RDB) is specified, the CRTSQLCBLI command also creates a package. If the ILE COBOL/400 compile is successful, the SQL precompiler automatically binds the SQL program and produces an access plan. The access plan for non-distributed SQL programs is stored in the program. The access plan for distributed SQL programs is stored with the package.

On CICS for OS/2 the precompile is performed with the DB2 PREP command. The precompile produces a .BND file for the program, which is

input to the bind process. You can use the DB2 BIND command to perform the bind, or you can set up VisualAge Generator Server environment variables to indicate that programs are to be bound the first time the program is run with the database. Refer to the *VisualAge Generator Server Guide for Workstation Platforms* documentation for a description of the environment variables associated with binding on OS/2.

On OS/2, AIX, Windows NT, HP-UX, and Solaris systems, the DB2 PREP command performs the precompile and bind. A .BND file is also produced, which can be used to bind the program to other databases.

ANSI standard static mode

If you specify the /ANSISQL COBOL generation option when generating a program to run in the MVS, OS/400, VSE, or VM environment, VisualAge Generator puts ANSI standard SQL statements in the generated COBOL program. Use ANSI standard SQL if you have a non-IBM ANSI database manager. Follow the preparation instructions for the database manager to prepare the generated program for the runtime environment.

Authorization considerations

An authorization identifier is required by SQL and the database manager to ensure that you have the authorization to perform operations on the database. Your authorization identifier is used for database authorization checking for all SQL functions in VisualAge Generator Developer. For run time, the authorization identifier used depends on whether a static or dynamic SQL statement is being run.

For all DB2 database managers other than DB2/400, the authorization identifier of the program developer performing the binding, DB2/VSE, or SQL/DS preprocessing is used for static SQL statements. The authorization identifier of the program user is used for dynamic SQL statements.

The preparation command file built by the generator from default command templates specifies that authorization checking should occur as described above (static SQL authorization checking is done using the ID of the owner or preparer of the program; dynamic checking is done using the ID of the user running the program).

By modifying the preparation templates you can specify that static SQL authorization be done with the ID the user running the program or dynamic SQL authorization be done with the ID of the person who prepared the program. Refer to the DB2/400 preparation command documentation for information on how to do this.

DB2 considerations

You can change the user ID used by DB2 for the authorization identifier using the security extensions in DB2. For more information, refer to a DB2 systems and database administration manual.

DB2/2 considerations

On OS/2 systems, you use the DB2/2 LOGON /L command to identify yourself to the database manager. If you have not logged on prior to running a DB2 program, the database manager prompts you for logon ID and password.

For server programs, the logon command must be issued on the server. All users run under the same authorization ID.

Additional DB2 considerations for VisualAge Generator Server

On OS/2, AIX, Windows NT, HP-UX, and Solaris systems, use the EZECONCT command in server and stand-alone programs to connect to DB2. The user ID and password parameters are honored. If no EZECONCT is issued at the time of the first SQL request, the program will connect using the user ID and password from the FCWDBUSER and FCWDBPASSWORD environment variables.

Using unqualified table names or synonyms

A table creator name is the authorization identifier of the person who created the table. If you do not fully qualify a table name in an SQL row record with the table creator name, the database manager supplies a default name.

If you are running a static SQL statement in a DB2 program, the default identifier is the value specified in the DB2 OWNER keyword during the bind or the value specified in the DB2/VSE or SQL/DS USERID option during DB2/VSE or SQL/DS preprocessing. Refer to the appropriate DB2, DB2/VSE, or SQL/DS manual for more information. If you are using dynamic SQL statements or any SQL functions in VisualAge Generator Developer, the default identifier is the user identifier of the user running the program or using VisualAge Generator Developer.

If DB2 or SQL/DS synonyms are used for table names, each developer that uses the SQL row record must create a synonym for each table name. If the SQL row record is used with dynamic SQL statements in a generated program, each user must create a synonym for each table. If the SQL row record is used only in static SQL statements in a generated program, only the plan owner as identified on the bind operation needs to create a synonym for the table name.

If you are running a static SQL statement in a workstation program, the default identifier is the authorization identifier of the user that bound the DB2

plan. Refer to the appropriate DB2 manual. If you are running a dynamic SQL statement within the program, the default identifier is the authorization identifier of the person running the program.

On DB2/400 systems, programs are always prepared with the *SQL naming convention. For static SQL statements, the default qualifier for table names is the collection name specified on the DFTRDBCOL parameter of the CRTSQLCBLI command used to prepare the program. If the parameter was not specified the default qualifier is the authorization ID of the user who owns (or prepared) the program. For dynamic SQL statements, the default qualifier is the authorization ID of the user running the program.

Database Considerations when using GUIs

Possible methods of database access when designing a GUI system include Smalltalk database parts, Java Data Access Beans, and VisualAge Generator programs. Results are unpredictable when you use more than one method in the same execution unit.

Accessing databases using DataJoiner

VisualAge Generator programs can access a variety of relational databases using IBM DataJoiner, such as the following:

- IBM DB2 on any platform
- Oracle Version 7 or later
- Sybase System10 or later

DataJoiner runs on AIX and Windows NT. VisualAge Generator workstation client programs can access DataJoiner using a client enabler product such as one of the following in place of a local DB2 database:

- IBM DataJoiner Client Support for AIX
- DataJoiner Client Application Enabler for UNIX systems
- DB2 Client Application Enabler for AIX
- DB2 Client Application Enabler for OS/2
- DB2 Client Application Enabler for DOS

You can access non-IBM database tables using the default SQL statements generated for VisualAge Generator SQL I/O options. User modified SQL statements are also likely to work correctly, but must be individually verified by the customer.

In addition, DataJoiner supports other functions that are likely to work with VisualAge Generator programs:

- Indirect access to a variety of other data sources accessed through:

  - ODBC, X/Open CLI compliant interfaces
  - IBI EDA/SQL
  - CrossAccess
- Read-only access to a collection of heterogeneous data sources as if it were one database.

You must verify the use of these functions in specific instances.

The information in the following sections describes DataJoiner unique considerations. See the earlier sections of the chapter for general SQL usage considerations. Refer to the DataJoiner documentation for more information on DataJoiner.

Getting Started

Experimenting with SQL using DataJoiner

If you are using relational databases for the first time, it is a good idea to completely understand what you can do with SQL by experimenting with the interactive facility provided with the DBMS. To experiment with SQL using DataJoiner, you can use the same tools as you would for DB2/6000 or DB2 for Windows NT.

Configuring remote data sources

Configuring a DataJoiner system requires the establishment of connections from clients to DataJoiner and from DataJoiner to the remote data sources. Client-to-DataJoiner connections are configured like client-to-DB2/6000 or client-to-DB2 for Windows NT connections. For specific details on configuring the DataJoiner-to-data source (DB2, Oracle, Sybase, and others) connections, refer to *DataJoiner Planning, Installation, and Configuration*

Your administrator must provide information about the remote data source and must associate the data source with a server name, as well as provide user ID and password information for remote users of the data source.

Using nicknames for table names

When running with DataJoiner, the SQL table name specified in a VisualAge Generator SQL row record can be either a three-part remote table name (server-name.remote-authorization-ID.table-name) or a DataJoiner nickname for the table name. Your administrator can use the DataJoiner CREATE NICKNAME statement to define nicknames to DataJoiner.

Using the PASSTHRU extension

You must use dynamic SQL to take advantage of the DataJoiner PASSTHRU extension. Use an SQLEXEC function with Execution Time Statement Build to issue SET PASSTHRU and SET PASSTHRU RESET statements plus any of the

SQL statements that a program issues while in PASSTHRU mode (any statements issued following a SET PASSTHRU and before a SET PASSTHRU RESET).

Accessing databases using ODBC

VisualAge Generator ODBC support provides access to relational and flat file systems on OS/2, Windows NT, HP-UX, AIX and Solaris. With ODBC, a VisualAge Generator program connects to a data source instead of directly to a database. A data source represents a connection between an ODBC driver and the DBMS. The mapping between a data source and an ODBC driver is managed by an ODBC driver manager.

Note: An ODBC driver manager and any necessary ODBC drivers must be available on the target or test platform. VisualAge Generator does not ship an ODBC driver manager or any ODBC drivers.

Within the same execution unit, do not mix VisualAge Generator programs generated with DBMS=ODBC, programs generated with DBMS=DB2, programs generated with DBMS=ORACLE, and/or VisualAge Smalltalk or VisualAge for Java database parts. The results in this case are unpredictable. DataJoiner is recommended for SQL programs which need to access a combination of database managers.

ODBC database drivers, including the ODBC driver manager, used with the VisualAge Generator ODBC support must be ODBC Version-2 compliant and at least level-1 conforming.

For authorization considerations, information on using unqualified table names or synonyms, or to experiment with SQL using ODBC, see the documentation provided with the ODBC software you are using or the database software you are accessing.

The information in the following sections describes ODBC unique considerations. See the earlier sections of the chapter for general SQL usage considerations.

Getting Started

Defining a data source

Before running a VisualAge Generator ODBC enabled program, the user needs to define to ODBC the data sources the program will access. When defining a data source, be aware that VisualAge Generator only supports a 1-18 character case-sensitive data source name.

Database setup

For certain database managers, there are administrative setup functions that need to be performed before using the VisualAge Generator ODBC support. Refer to the documentation provided for the database driver that you are using for more specific information about setup that is required for that database driver.

VisualAge Generator Server setup

In order to run generated C++ ODBC programs using VisualAge Generator, some setup of the VisualAge Generator Server ODBC modules may be required. Refer to the *VisualAge Generator Server Guide for Workstation Platforms* documentation for setup information.

Using ODBC in VisualAge Generator

When using ODBC, the database manager **must be started manually** before the first connection is made to a data source.

The Cursor WITH HOLD option is not supported under the current VisualAge Generator ODBC support.

Defining and testing ODBC programs

To use the ODBC interface from VisualAge Generator Developer, go to the **VisualAge Generator Options/Preferences** and select **SQL**. Then specify the ODBC data source name in the **Database name** field and select ODBC for the **Database management system**. The default database management system is DB2.

Validating and generating ODBC programs

To validate or generate an SQL program for ODBC, use the /DBMS=ODBC generation option. The /DBMS=DB2 option generates native DB2 embedded SQL programs. The /DBMS=ORACLE option generates native Oracle embedded SQL programs. If the /DBMS option is not specified, the **Database management system** value specified in the VisualAge Generator SQL Options/Preferences is used. You may also select **Database management system** on the Validation page of the Generation Options notebook.

For ODBC enabled programs, there is no DB2 or Oracle precompilation phase during the preparation process, and no bind files are created with the program module.

The results of validation of SQL statements via ODBC is dependent on the ODBC driver implementation of the SQLPrepare ODBC API. This API is used by VisualAge Generator to validate SQL statements when using ODBC. However, according to the ODBC 2.0 standard (which VisualAge Generatorconforms to), syntax checking of the SQL statement does not have to be done by the ODBC driver during a call to this API function. Some drivers

do syntax checking and return errors on a call to this API while some defer it until a corresponding SQLExecute API call is made.

When using the DB2 ODBC driver, syntax checking can be turned on by setting the DEFERREDPREPARE environment variable to 0. The Oracle ODBC driver defers all syntax checking until the SQLExecute API. There is no way to alter this behavior.

Check with your ODBC vendor to determine the behavior of the SQLPrepare ODBC API in regards to syntax checking for your ODBC driver.

Running ODBC programs

To run ODBC enabled C++ programs, specify the ODBC Data Source Name, in place of the actual database name, using either the FCWDBNAME_<applname> or EZERSQLDB environment variables. FCWDBNAME_<applname> specifies the name of the data source name to be used for running a specific program. This environment variable allows the data source name to be specified on a program-by-program basis. If FCWDBNAME_<applname> is not specified, EZERSQLDB is used.

SQL syntax

You can use any of the default SQL statements generated for VisualAge Generator SQL I/O options to access databases through the ODBC interface. For maximum portability, user modified SQL statements should use the standard syntax for SQL statements as defined in the Microsoft ODBC Programmer's Reference and SDK Guide.

Data type considerations

It is strongly recommended that you first define the SQL table using the appropriate DBMS before you define the VisualAge Generator SQL row records. Then use the record editor's Retrieve SQL function to define the corresponding VisualAge Generator SQL row record data items. Alternately, you may define the SQL row record manually and specify the SQL column data types using the DB2 native SQL data codes. For information on how VisualAge Generator converts Oracle data types to VisualAge Generator data types, see Table 15 on page 76.

VisualAge Generator ODBC support uses a character data area to set and retrieve SQL date/time/timestamp columns defined in the ODBC data source.

When retrieving a date column from the data source, the date SQL data is converted to character data. The resulting string is in ″yyyy-mm-dd″ format. When retrieving a time column from the data source, the time SQL data is converted to character data. The resulting string is in ″hh:mm:ss″ format.

When setting a date column in the data source, the character data area should contain date data in the format defined in the data source. When setting a

time column in the data source, the character data area should contain time data in the format defined in the data source.

It should be noted that Oracle Date data type columns contain both date and time information and the corresponding VisualAge Generator character data item must be at least 19 bytes to accommodate the data returned. Otherwise, the program will terminate on retrieval of the column with a numeric overflow error.

DB2 SQL data types graphic, vargraphic, and long vargraphic are not supported by the ODBC drivers. VisualAge Generator programs that access a DBCS data item which maps to one of these data types will cause run time errors when running under ODBC.

Testing results of an SQL I/O option for ODBC

It is strongly recommended that VisualAge Generator programs test the SQL row record for the VisualAge Generator record I/O error values to determine if an I/O operation completed successfully. When the record state indicates an error condition was encountered, additional information sometimes may be obtained by examining the SQLSTATE in the EZESQLCA structure.

For more information, see "Testing the results of an SQL I/O option" on page 54.

The values returned in EZESQCOD and SQLSTATE vary between different database managers. After accessing a data source using ODBC, EZESQCOD usually contains the native database manager return code. The result of comparing EZESQCOD for a specific database manager with other database managers is unpredictable.

I/O error values

ODBC drivers return diagnostic error information using the SQLSTATE variable. The ODBC driver manager may also return error information via return codes. The HRD status is set when the value of SQLSTATE is ″40001″, ″23505″, or ″23000″ or whenever the driver manager returns a return code of -1 (SQL_ERROR), -2 (SQL_INVALID_HANDLE), or 2 (SQL_STILL_EXECUTING). Additionally, UNQ status is set when the value of SQLSTATE is ″23505″ or ″23000″. DED status is set when the value of SQLSTATE is ″40001″. ERR status is set when the driver manager returns any positive return code except 100 (SQL_NO_DATA_FOUND).

Special function words

The following considerations apply to the VisualAge Generator special function words when using ODBC.

EZESQISL

The isolation level value should be set prior to a data source connection. Each connection can assume one of the following values:

0 The isolation level is set to repeatable read.

1 The isolation level is set to cursor stability.

This special function word is not currently supported by the VisualAge Generator Test Facility.

EZESQLCA

The length of the communication area remains 136 bytes long. The last 5 bytes of the area contains the SQLSTATE associated with an ODBC I/O error.

SQLSTATE contains a 5-character warning or error code value that is defined in the SQL92 standard, unlike SQLCODE (EZESQCOD) values which are not consistent from product to product. The ODBC SQLSTATE error code is more general than the SQLSTATE returned by a specific database manager.

Only the SQLCODE, SQLWARN1 and SQLSTATE fields are set in the EZESQLCA. All other fields contain the initial value and are not set by any ODBC I/O operation.

A VisualAge Generator sample working storage record named HptSQLCA, which maps to the DB2 version of SQLCA, is shipped with the VisualAge Generator Developer product, in the HptSampleEzerempParts package/application in the ezeremp.dat file.

EZESQRD3

Contains initial value and is not set by any ODBC I/O operation.

EZESQRRM

Contains initial value and is not set by any ODBC I/O operation.

EZESQWN1

Contains a 'W' if the last ODBC I/O operation causes the DBM to truncate character data because of insufficient space in a program host variable.

EZESQWN6

Contains initial value and is not set by any ODBC I/O operation.

EZECOMIT/EZEROLLB

On COMMIT or ROLLBACK operations, all open cursors are closed.

Cursor WITH HOLD option is not supported under the current VisualAge Generator ODBC support.

Accessing distributed databases using EZECONCT

Using the call EZECONCT statement a VisualAge Generator program can connect to one or more data sources, depending on the DBMS. For example, Oracle supports multiple connections per process while DB2 supports only one connection.

Each ODBC connection establishes a separate logical unit of work and unlike the distributed unit of work, there is no coordination between the SQL I/O statements that are executed on different connections. When a commit or rollback is triggered, VisualAge Generator attempts to perform commit/rollback on all connected data sources. If an error occurs, the data sources might be left in an inconsistent state. DataJoiner is recommended for programs that need to update more than one database.

Accessing databases using Oracle

VisualAge Generator Oracle support provides access to Oracle databases on OS/2, AIX, CICS for AIX, Windows NT, CICS for Windows NT, HP-UX, Solaris, and CICS for Solaris.

Note: Oracle7 is not supported on Solaris and Oracle8 is not supported on OS/2.

Within the same execution unit, do not mix VisualAge Generator programs generated with DBMS=ODBC, programs generated with DBMS=DB2, programs generated with DBMS=ORACLE, and/or VisualAge Smalltalk or VisualAge for Java database parts. The results in this case are unpredictable. DataJoiner is recommended for SQL programs which need to access a combination of database managers.

The information in the following sections describes Oracle unique considerations. See the earlier sections of the chapter for general SQL usage considerations.

Getting started

Experimenting with SQL using Oracle

If you are using relational databases for the first time, it is a good idea to completely understand what you can do with SQL by experimenting with the interactive facility provided with the DBMS. For Oracle, you can use the SQL* Plus utility shipped with the Oracle software. See the documentation provided with the Oracle software for more information.

VisualAge Generator Setup

Before generating SQL programs for Oracle, make sure that the environment variables specified in the VisualAge Generator Server Guide have been set for the appropriate environment.

Database setup

Refer to the appropriate Oracle documentation for information on creating or configuring databases.

VisualAge Generator Server setup

In order to run generated C++ Oracle programs using VisualAge Generator Oracle7 modules, some setup of the VisualAge Generator Server Oracle modules may be required. Refer to the *VisualAge Generator Server Guide for Workstation Platforms* documentation for setup information.

Using Oracle in VisualAge Generator

Defining and testing Oracle programs

Oracle databases are not directly accessible via the embedded SQL interface from the VisualAge Generator Definition Facility or the Test Facility. The ODBC interface must be used for defining and testing programs accessing Oracle databases. Once these programs have been thoroughly tested, they can be generated to use the Oracle embedded SQL interface for Oracle on the supported execution platforms. For information on the supported environments, see "Accessing databases using Oracle" on page 74.

For information on ODBC support, see "Using ODBC in VisualAge Generator" on page 70.

Validating and generating Oracle programs

To validate or generate an SQL program that accesses Oracle, use the /DBMS=ORACLE generation option. If the /DBMS option is not specified, the Database management system value specified in the VisualAge Generator SQL Options/Preferences is used.

Running Oracle programs

To run VisualAge Generator programs generated to access Oracle databases, specify the Oracle database name using either the FCWDBNAME_<progname> or EZERSQLDB environment variables. FCWDBNAME_<progname> specifies the name of the Oracle database to be used for running a specific VisualAge Generator program. This environment variable allows the database name to be specified on a program-by-program basis. If FCWDBNAME_ <progname> is not specified, EZERSQLDB is used.

Data Type Considerations

It is strongly recommended that you use the VisualAge Generator's Retrieve SQL function for dynamically creating the SQL row record. Using the Retrieve SQL function insures that your SQL row record data item definitions match

the columns in the SQL table object. First, define the SQL table objects using the appropriate tool for your Database Management System. Then use the record editor's Retrieve SQL function to define the corresponding VisualAge Generator SQL row record data items. Alternatively, you can define the SQL row record manually and specify the SQL column data types using the DB2 native SQL data codes. For more information on supported ANSI data types, refer to the Oracle Server SQL Reference Guide.

Using the Retrieve SQL function, VisualAge Generator converts the Oracle data types to the VisualAge Generator data types as shown in Table 15.

Table 15. Oracle to VisualAge Generator Data Type Conversion

Oracle Data Type	VisualAge Generator Data Type	VisualAge Generator SQL Code
CHAR	Char	453
DATE	Char	453
LONG	Char	457
LONG RAW	Hex	481
NUMBER	Hex	481
NUMBER(p,s)	Pack	485
RAW	Hex	481
VARCHAR2	Char	449

Note: Oracle data types that are not listed are converted to the VisualAge Generator Hex data type with an SQL code of 481.

Date Data Type

The DATE data type in Oracle contains both the date and time information. The corresponding VisualAge Generator character data item must be at least 19 bytes to accommodate the data returned. Oracle does not provide direct support for the TIME or TIMESTAMP date type. You can use the VisualAge Generator string functions to substring for the time data.

The default date format is specified by the Oracle server initialization parameter NLS_DATE_FORMAT and is usually a string such as 'DD-MON-YY'. This default can be changed on the server by specifying another format for this initialization parameter or overridden on the client machine by specifying the desired format with the following environment variables:

```
NLS_LANG
NLS_DATE_FORMAT
```

For example, to tell Oracle to return the date in the format of 'YYYY/MM/DD' the following system environment variables must be set for Windows NT:

```
NLS_LANG=american_america
NLS_DATE_FORMAT=YYYY/MM/DD
```

Testing results of an SQL I/O option for Oracle

The values returned in EZESQCOD and SQLSTATE vary between different database managers. It is strongly recommended that VisualAge Generator programs test the SQL row record for the VisualAge Generator record I/O error values to determine if an I/O operation completed successfully.

For more information, see "Testing the results of an SQL I/O option" on page 54.

I/O error values

Oracle returns diagnostic error information much like DB2 does in the SQLCODE variable of the SQLCA. The HRD status is set for all negative SQL codes. Additionally, UNQ status is set for SQL return code -1 (insert or update was not successful because of a duplicate index column value). DED status is set for SQL code of -60. ERR status is set for any positive return code except 100 (NRF).

Special function words

The following considerations apply to the VisualAge Generator special function words when using Oracle.

EZESQCOD
The values contained in EZESQCOD are documented in the Oracle Server Messages and Codes Manual.

EZESQISL
This special function word is not used by Oracle.

EZESQLCA
The length of the communications area is 136 bytes. The format differs slightly from the DB2 SQLCA. Refer to the Oracle SQL publications for more information on the format of the SQLCA.

EZESQRD3
Contains the actual number of rows affected after an INSERT, UPDATE, or DELETE operation.

EZESQRRM
Contains the message text corresponding to the error code stored in EZESQCOD.

EZESQWN1
Contains a 'W' if the last Oracle I/O operation caused a character data item to be truncated.

EZESQWN6
This special function word is not set by Oracle.

EZECOMIT/EZEROLLB

EZECONCT
When specifying the UOW parameter, the following considerations apply:

Only Type 1 Connects (R) are supported. Only one database can be connected at a time. EZECOMIT or EZEROLLB must be issued prior to connection to another database. Connection to another database ends an existing connection. All cursors are closed when the new connection occurs.

A value of DISC or SET is ignored.

If DCURRENT or DALL is specified, an EXEC SQL ROLLBACK WORK RELEASE is issued.

Authorization considerations

An authorization (userid and password) is required for precompilation on Windows NT and OS/2. It is optional on all other systems.

Using unqualified table names or synonyms

Each user in an Oracle database has an associated schema name of the same name as the user. If a table name, view name, snapshot name or synonym is not fully qualified in an SQL row record or SQL statement, the default value for the schema name is the name of the user currently connected to the database.

Accessing DB2/MVS stored procedures

Transforming new or existing VisualAge Generator client/server programs into DB2 stored procedures for DB2/MVS can be done with little effort. The following sample programs and related files were created to guide you through this process. The source code for the sample files are included in the SAMPLES sub-directory where the VisualAge Generator Developer product is installed. The files and what is included in each file are:

STAFFJ.DAT and STAFFS.DAT
STFPROC, the stored procedure which runs on DB2/MVS and, ACALLSP, the calling program that invokes the stored procedure. STAFFJ.DAT is for VisualAge Generator for Java and STAFFS.DAT is for VisualAge Generator Smalltalk. For information on how to import the parts from these files into your Repository/ENVY library, see the *VisualAge Generator Getting Started* documentation.

STAFF.QMF
STAFF table in QMF format. To upload the file to the host using IBM's eNetwork Personal Communications, issue the following command:

```
send staff.qmf b:qmf.staff [crlf recfm(f) lrecl(44)
```

STFPROC.SQL
SQL statements used to catalog the stored procedure in the DB2 system catalogs.

Defining stored procedures

A stored procedure is defined as a called batch program that accesses only relational or DL/I databases, not files. The stored procedure cannot XFER to another program. The only other special considerations for defining stored procedures are in defining the parameter list. The following is a list of the special considerations for defining the parameter list for a stored procedure:

- Parameters can be individual data items up to 254 bytes long or records up to 32K bytes long.
- Individual data item types are restricted to the types supported by SQL row records: Char, DBCS, Bin, Pack and Hex.
- Record definitions must be defined with a top-level character item that includes all of the other items in the record. The top-level item is specified as the parameter in the SQL CALL coded in the calling program.
- Records up to 254 bytes long can be passed as fixed-length SQL CHAR parameters.
- Records greater than 254 bytes in length must be passed as SQL VARCHAR parameters, and a second record part definition must be defined for use in the called parameter list with a 2-byte Bin data item added to the front of the record structure.

Defining the stored procedure call

To define the stored procedure call, do the following:

1. Use an SQLEXEC process to define the SQL CALL statement to call the stored procedure, specifying the individual data item or top-level record item for each parameter as a host variable on the SQL CALL.
2. Since DB2 does not know about the structure of record parameters, call EZECONV for each record parameter before the call to convert the parameter to host data format on the way to the stored procedure, and after the call to convert the parameter back to client format.
3. Specify individual items or top-level record items of up to 254 bytes long as host variables on the SQL CALL statement.
4. For records greater than 254 bytes, define an SQL record item with the same length as the record and with SQL data code = 457 (VARCHAR). Specify the SQL item as the host variable and move the record contents

from the record to the host variable prior to the call, and back from the host variable to the record after the call.

Preparing a VisualAge Generator stored procedure

You prepare a VisualAge Generator stored procedure as you would prepare any other MVS BATCH called program. The program should be link-edited with the Call Attachment Facility language interface module, DSNALI. The following template, EFK2MPBC.TPL, was permanently changed to use the CAF interface module.

```
//*****************************************************************
//** EFK2MPBC - PREPARE MVSBATCH APPLICATION WITH DB2 ACCESS
//** AND NO DLI ACCESS
//** DB2 PRECOMPILE, COMPILE, LINK AND BIND
//*****************************************************************
//PCLB EXEC ELAPCLB,MBR=%EZEMBR%,ENV=%EZEENV%,DATA=%EZEDATA%,
//CGHLQ='%EZEPID%'
//L.SYSIN DD *
CHANGE ELAAPPL(%EZEMBR%)
INCLUDE SELALMD(ELARMAIN)
INCLUDE SELALMD(ELARSINT)
INCLUDE SELALMD(ELASTB07)
INCLUDE SYSLIB(DSNALI)
ENTRY %EZEENTRY%
NAME %EZEMBR%(R)
/*
//B.SYSTSIN DD DISP=SHR DSN=%EZEPID%.%EZEENV%.EZEBIND(%EZEMBR%)
```

Declaring stored procedures

All stored procedures must be defined in the DB2 system table SYSIBM.SYSPROCEDURES. The following information was cataloged for this example of a VisualAge Generator stored procedure.

```
INSERT INTO SYSIBM.SYSPROCEDURES
(PROCEDURE, AUTHID, LUNAME, LOADMOD, COLLID, LINKAGE,
LANGUAGE, RUNOPTS, STAYRESIDENT, IBMREQD,
PARMLIST)
VALUES('STFPROC', ' ', ' ', 'STFPROC', 'STFPROC', ' ',
'COBOL', ' ', 'Y', 'N',
'STFPARM SMALLINT INOUT,
VARCHAR(1660) FOR BIT DATA INOUT')
```

Parameter list data types

The parameters passed by DB2 to the stored procedures can be in several forms (using SQLDA, host variables, constants, and NULL). Currently, VisualAge Generator stored procedures support passing parameters only as host variables. The host variables for records and hex data items should be defined as binary data (FOR BIT DATA) when defining the parameters in the DB2 system catalog. This prevents DB2 from performing conversion on the data being passed.

Parameter size

When calculating the size of a VARCHAR parameter, do not include the length of the additional 2-byte field in the size. You will notice that the parameter STFPROC_REQ_MSG was defined with a length of 1660 to DB2. The extra 2 bytes are inserted by DB2 when the parameter is passed up.

Defining DB2 linkage conventions

Parameters for VisualAge Generator stored procedures should be defined as INOUT (input/output) parameters. INOUT means that data will be flowing to and from the stored procedure. The sample called program was tested with both SIMPLE and SIMPLE WITH NULLS. The SIMPLE linkage convention was the only parameter list convention that worked successfully for VisualAge Generator. To specify the SIMPLE linkage convention, enter a blank value for the LINKAGE column when the stored procedure is defined to DB2.

Binding the stored procedure package

The following bind command was used to create the package for the sample VisualAge Generator stored procedure, STFPROC. If your program calls other generated programs using the VisualAge Generator CALL or DXFR statement, include the DBRMs for these programs in the stored procedure package.

```
DSN SYSTEM(DSNE)
* EFK2MBDD
* BIND TSO APPLICATION WITH DB2 ACCESS AND NO DLI ACCESS
* BIND MVSBATCH APPLICATION WITH DB2 ACCESS AND NO DLI ACCESS
BIND PACKAGE(STFPROC) -
MEMBER(STFPROC) -
ACT(REP) -
VALIDATE(BIND) -
ISOLATION(CS)
* OWNER(OWNERGRP)
BIND PACKAGE(STFPROC) -
MEMBER(ELADBRM4) -
ACT(REP) -
VALIDATE(BIND) -
ISOLATION(CS)
* OWNER(OWNERGRP)
```

Calling a VisualAge Generator stored procedure

Stored procedures are invoked using the SQL statement CALL. The program, ACALLSP, contains an example of how to call a stored procedure. The program was generated to run as a C++ program in the Windows NT environment. The steps to prepare ACALLSP to call stored procedures, STFPROC, are discussed later in more detail.

Defining host variables

1. Create host variables for each parameter passed to the stored procedure.
2. Define host variables corresponding to working storage records greater than 254 bytes long as VARCHAR, data code 457.

3. Specify individual items or top-level record items of up to 254 bytes long as host variables on the SQL CALL statement.
4. Use the default SQL data code for the item type for host variables that are less than or equal to 254 bytes long.

For our sample, two host variables that correspond to the two parameters were created, TEST_REC.STARTING_ID and TEST_REC.STFLIST_DATA. Both variables were defined in the same SQL row record. We could have defined an SQL row record for each parameter.

Invoking the stored procedure

Invoke the stored procedure (as shown below) from an SQLEXEC process, using the SQL CALL statement, passing the host variables in the order expected by the called program.

```
CALL STFPROC
  (:TEST_REC.STARTING_ID,
   :TEST_REC.STFLIST_DATA)
```

Converting data

Because DB2 does not understand how to work with VisualAge Generator data structures, the record parameter is defined as bit data to DB2. As a result, the calling program is responsible for converting the data before and after calling the stored procedure. The following is an example of how this was coded in the main process, PCALLSP_MAIN.

```
EZEFEC = 1;
PSHOW_MSTFMN();
  WHILE EZEAID NOT PF3;
    STFLIST_SEARCH_ROW.STARTING_ID = MSTFMN.STARTING_ID;
    TEST_REC.STARTING_ID = STFLIST_SEARCH_ROW.STARTING_ID;
    EZECONV( STFLIST_REQ_MSG2,'L','ELACNENU');
    PCALL_STORED_PROC();
  ;   /* CALL STFLIST  STFLIST_SEARCH_ROW, STFLIST_REQ_MSG2;
    IF EZESQCOD NE 0;
      MOVE EZESQCOD TO MSTFMN.STAFFIDX_WS[1];
      EZEREPLY = 0;
      EZEROLLB();
    END;
    STFLIST_REQ_MSG2.STFLIST_DATA = TEST_REC.STFLIST_DATA;
    EZECONV( STFLIST_REQ_MSG2,'R','ELACNENU');
    MOVEA STFLIST_REQ_MSG2.STAFFIDX_WS TO MSTFMN.STAFFIDX_WS FOR 10;
    MOVEA STFLIST_REQ_MSG2.NAME_WS TO MSTFMN.NAME_WS FOR 10;
    MOVEA STFLIST_REQ_MSG2.SALARY_WS TO MSTFMN.SALARY_WS FOR 10;
    MOVEA STFLIST_REQ_MSG2.COMM_WS TO MSTFMN.COMM_WS FOR 10;
    PSHOW_MSTFMN();
  END;
```

Testing stored procedures

The dynamic SQL interface used by the Test Facility does not support calling a stored procedure. SQL error -104 is returned if a call is made to a stored procedure from the Test Facility. The best way to test a stored procedure is to

test it as a normal client/server program using the CALL statement. The following list describes what we did to test our stored procedure:

- Commented out the SQLEXEC process containing the DB2 SQL statement CALL and replaced it with a VisualAge Generator CALL
- Commented out the calls to the conversion routines
- Commented out the statements that moved the data back and forth between the working storage records and the host variables
- Removed the VARCHAR length variable from the parameters on the called parameter list.

When you are ready to generate the called program for MVS BATCH, reverse the steps above. Be sure to insert the VARCHAR length variable back into the appropriate working storage record. For this example, record STFPROC_REQ_MSG was the only record that required the VARCHAR length variable.

After the stored procedure is prepared and ready to start, generate and test the calling program.

Tracing and debugging

To debug your VisualAge Generator client/server program, set the environment variable CSO_DUMP_CONV=YES to show the conversion of data before and after the call to the stored procedure as follows:

```
SET CSO_DUMP_CONV=YES
```

Chapter 3. Developing DL/I programs

You can use VisualAge Generator Developer to define Data Language I (DL/I) programs. A generated DL/I program can be run on one of the following systems: CICS for MVS/ESA, CICS for VSE/ESA, IMS/VS, IMS BMP, MVS/TSO, MVS batch, and VSE batch. The differences between these systems are noted throughout this chapter.

In a DL/I program environment, there are two primary responsibilities:
- Defining the databases
- Developing the programs that access the databases

The database definition function is usually assigned to a database administrator. The database administrator does the following tasks for VisualAge Generator DL/I programs. These tasks are the same for programs written in other programming languages:
- Definition of databases
- Definition of Program Specification Blocks (PSBs)
- Definition of databases in the CICS for MVS/ESA or CICS for VSE/ESA file control table
- Definition of the database and PSB directory (CICS for MVS/ESA only)
- Definition of the valid PSBs, among other specifications, in the Application Control table (CICS for VSE/ESA only)
- Definition of the databases and valid PSBs in the IMS system definition (IMS only)

Most installations choose to have the database administrator control and maintain a single common definition for all database segments in the Repository/ENVY library. When a new program is designed, the database administrator defines each PSB and each segment in the PSB for that program in the Repository/ENVY library. Using the Repository/ENVY library definitions, VisualAge Generator creates the DL/I parameter lists that enable the program to access the databases defined in the PSB

Definition of DL/I programs is similar to the definition of other VisualAge Generator programs. You can use many of the same VisualAge Generator I/O options you use for a file-based program and specify DL/I segments as I/O objects. VisualAge Generator creates the DL/I call needed to run the I/O option.

Introduction to DL/I

DL/I is a data management control system that enables you to create, access, and maintain large, shared databases. VisualAge Generator helps you quickly develop programs that access DL/I databases.

Note: DL/I calls are also used in IMS to handle terminal and program-to-program communications. This chapter describes only database calls. For more information about calls used in the IMS environment, refer to "Chapter 5. Developing IMS programs" on page 137.

You need to understand the following basic DL/I concepts before you begin developing a DL/I program:

Segments
The primary unit of data in a DL/I database is the segment. A segment is similar to a record. It is a single block of data divided into data fields that are similar to the data items in a record.

Database Hierarchy
A single database can contain many types of segments. These segments are arranged in a hierarchical (top down) relationship. The segment at the top of the hierarchy is called a root segment. Each segment can have one or more dependent segments related to it at a lower level in the hierarchy. A segment with a dependent segment is called the parent of the dependent segment. The dependent segment is called a child segment. Each segment in the database, except for a root segment, has one and only one parent. The root segment has no parent.

Sequence Field
Each segment type in a database can have one of its fields designated as a sequence field. The value of the sequence field determines the order that the segments are stored and retrieved from the database. When a parent segment has multiple occurrences of the same child segment, those child segments are stored in sequence field order under that parent, if a sequence field is defined for the child segment.

Program Specification Block (PSB)
A PSB is a formal DL/I description of the hierarchical database structures that a program can access. VisualAge Generator supports the definition of a Repository/ENVY library part that contains a subset of the information in the DL/I PSB. The PSB shows the hierarchical relationships that exist between types of segments.

Program Communication Block (PCB)
A PCB is an entry in a PSB. Each database PCB describes one hierarchical data structure that a program can use. The data structure

might correspond directly to the structure of a physical or logical DL/I database or might invert the database structure through access by a secondary index.

DL/I Call

A DL/I call is an invocation of DL/I by a program. The parameter list passed for a database call provides DL/I with the following information:

Function Code

Indicates if DL/I is to get, insert, replace, or delete segments from the database.

Database identifier

Points to the Program Communication Block (PCB) that identifies the database that DL/I is to access on the call.

I/O Area Address

Identifies the address of the buffer that contains the segment after it is read from the database or before it is written to the database.

Segment Search Argument (SSA) List

Lists a set of search criteria that enables DL/I to select the segments that it retrieves from the database or specify the position of segments it inserts into the database.

When you code a DL/I program in a language like COBOL or PL/I, you either code the DL/I parameter list directly or use the command level interface of CICS for MVS/ESA or CICS for VSE/ESA to create the DL/I parameter list.

VisualAge Generator creates DL/I parameter lists for you based on the I/O option and the position of the I/O object in the PSB. You can view the DL/I call created for the function. You can also modify the DL/I call to use additional DL/I functions.

Database Position

When a program is running, DL/I maintains a position pointer for each PCB in the program PSB. The pointer indicates the place in the database where a SCAN function (DL/I get next function) begins searching for the segment to retrieve.

The position pointer is set on any successful DL/I call to point to the segment following the last segment accessed on the call. If no calls are issued, the current position indicates the start of the database. If the end of database condition is encountered, the current position becomes the start of the database.

As DL/I continues scanning a database for a segment that satisfies the SSA list criteria, DL/I accesses each root segment in the order it appears in the database. When DL/I finds a root segment, it accesses all the dependents of the root before scanning the next root. As DL/I scans the dependent segments, it first tries to read the next segment at the next lower level. If there is not a lower level, it reads the next segment at the same level. If there are no more segments at the current level, it returns to the previous level to search for the next segment. This process is called the "top to bottom, left to right" search order.

Example: customer database

Figure 6 shows the structure of a customer database used in the examples throughout this chapter. The database contains information needed for processing customer orders, such as customer name, address, current order information, and credit status.

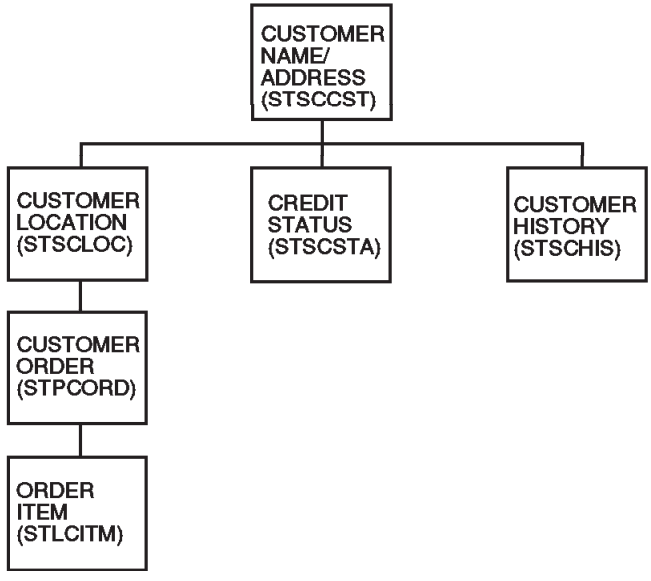

Figure 6. An example of a customer database

There are 6 types of segments in the customer database:

Customer Name and Address Segment (STSCCST)
This is the root segment in the database. There is one of these segments for each customer. The key field is the 6-byte customer number. The following fields are in the segment:

NAME	*DESCRIPTION*	*LENGTH IN BYTES*

STQCCNO	Customer number	6
STUCCNM	Customer name	25
STQCCA1	Customer address line 1	25
STQCCA2	Customer address line 2	25
STQCCA3	Customer address line 3	25

Customer Location Segment (STSCLOC)

Each customer can have multiple ship locations. Each location is represented by a unique location segment chained off the single customer segment. The location segment is a child segment of the customer segment. The segment sequence field is a location number that is unique for that customer. The location segment also contains a location name and address.

NAME	*DESCRIPTION*	*LENGTH IN BYTES*
STQCLNO	Location number	6
STFCLNM	Location name	25
STFCLA1	Location address line 1	25
STFCLA2	Location address line 2	25
STFCLA3	Location address line 3	25

Customer Order Segment (STPCORD)

Each location can have any number of active orders including 0. The order segment is a child of the location segment. The orders are uniquely identified by an order number and date. Each order also contains some reference data, an item count, and a total order amount.

NAME	*DESCRIPTION*	*LENGTH IN BYTES*
STQCODN	Order date/number	12
STFCORF	Order reference data	25
STFCOIC	Order item count	6
STFCOAM	Order amount	12

Order Item Segment (STLCITM)

Multiple kinds of items can be included in each order. There is one item segment for each item in the order. Each item is identified by an inventory item number and a line item number. For this order, the segment also contains the quantity ordered, the quantity shipped, the quantity back ordered, and the amount charged for the item.

The segment, as presented to the program, also contains an item description, total quantity on hand, total quantity on order for all customers, quantity reserved, unit price, and unit of issue. This last set of information is physically stored only once in a separate

inventory database. DL/I extracts the information from the inventory database using the item number in the order item segment as a logical link to the inventory database. From the program's point of view, this information is presented as if it was part of the order item segment.

NAME	*DESCRIPTION*	*LENGTH IN BYTES*
STKCIIN	Inventory item number	6
STQCILI	Line item number	2
STFCIQO	Quantity ordered	6
STFCIQS	Quantity shipped	6
STFCIQB	Quantity back ordered	6
STFCIAM	Item amount	12
STQIINO	Item number	6
STFIIDS	Description	25
STFIIQH	Quantity on hand	6
STFIIQO	Quantity on order	6
STFIIQR	Quantity reserved	6
STFIIPR	Unit price	6
STFIIUN	Unit of issue	1

Credit Status Segment (STSCSTA)

This segment contains information about the credit status of the customer and is a child of the customer segment. There is only one credit segment per customer, so you do not need a key field to uniquely identify the credit segment. The information is placed in a separate segment to permit access to only authorized users. A segment contains 2 fields: credit limit and credit balance.

NAME	*DESCRIPTION*	*LENGTH IN BYTES*
STFCSCL	Credit Limit	12
STFCSBL	Credit balance	12

Customer History Segment (STSCHIS)

The history segment contains summary information about closed orders. There is 1 segment per closed order. The closed orders for a specific customer are chained off the customer record in date and order number sequence. This segment is defined with a variable length to provide flexibility in storing order status information. The first field in the segment is the halfword containing the segment length.

NAME	*DESCRIPTION*	*LENGTH IN BYTES*
STGCSL	Segment length	2

STQCHDN	Order date/number	12
STFCHRF	Order reference data	25
STFCHIC	Order item count	2
STQCHAM	Order amount	12
STQCLOS	Order status	77

Defining a sample PSB

The database administrator defines the database structures that a program can use in a DL/I program specification block. Figure 7 shows the PSB for a sample print program. The customer database is the second PCB entry in the PSB. Note that each segment in the database is represented by a SENSEG statement in the PCB entry. Each SENSEG segment identifies the segment and its parent segment by name.

```
TITLE 'DL/I SAMPLE PRINT PROGRAM - PSB'
PCB TYPE=DB,DBDNAME=STDIDBL,PROCOPT=G,KEYLEN=50,POS=S,
    PROCSEQ=STXININ
SENSEG NAME=STPIITM,PARENT=0
SENSEG NAME=STLICOR,PARENT=STPIITM
SENSEG NAME=STSIVND,PARENT=STPIITM
SENSEG NAME=STLISUB,PARENT=STPIITM
SENSEG NAME=STSILOC,PARENT=STPIITM
PCB TYPE=DB,DBDNAME=STDCDBL,PROCOPT=AP,KEYLEN=50,POS=S
SENSEG NAME=STSCCST,PARENT=0
SENSEG NAME=STSCLOC,PARENT=STSCCST
SENSEG NAME=STPCORD,PARENT=STSCLOC
SENSEG NAME=STLCITM,PARENT=STPCORD
SENSEG NAME=STSCSTA,PARENT=STSCCST
SENSEG NAME=STSCHIS,PARENT=STSCCST
PSBGEN LANG=ASSEM,PSBNAME=STBICLG
END
```

Figure 7. Sample PSB for Customer Database Program

Note: On MVS/TSO, MVS batch, IMS/VS, and IMS BMP systems, you must include the parameter CMPAT=YES on the PSBGEN statement.

The customer database and the sample PSB are used in the following sections as examples in defining databases to VisualAge Generator and in defining a program that refers to the databases.

Defining DL/I data

Defining DL/I databases through the PSB definition facility

The PSB definition facility enables you to define a set of DL/I databases that a program can access. VisualAge Generator uses the PSB definition to create and validate DL/I calls. The PSB definition is stored as a separate part in the

Repository/ENVY library. The part is called a PSB definition because it contains a subset of the information in the original DL/I PSB.

To define a PSB part, define a PCB for each PCB statement in the DL/I PSB. For each database PCB, enter the following:
- PCB type (DB)
- Database name
- Name of index key field, if database is to be accessed by a secondary index (PROCSEQ operand specified on the PCB statement in the DL/I PSB)

For each segment in the database (SENSEG statements in the DL/I PSB following the PCB statement), enter the following:
- Segment name
- Parent segment name (blank for root segment)

The segment names in the PSB part must be the same as the segment names in the DL/I PSB. The secondary index key name in the PSB part must be the name specified on the XDFLD statement that defines the secondary index field in the DL/I database description.

To use the PSB definition with a program, you must do the following:
1. Enter the PSB name in the program specifications.
2. Define each segment in the PSB as a record with DL/I segment organization.
3. Define each secondary index key as a data item. The item must have the same length as the DL/I secondary index key field.

Note: If you plan to use the PSB in an IMS program, you must also define PCBs for issuing IMS teleprocessing DL/I calls to terminals and for accessing serial files as GSAM files or IMS message queues. PSBs that include the IMS PCB definitions can also be used in the other DL/I environments. See "Sharing a function when programs use different PSBs" on page 100 for more information about defining PSBs.

Segment record definition

You define DL/I segments using the record editor. Give the record the same name that the segment has in the PSB definition.

During record specification you define the record organization as a DL/I segment. If the segment is sequenced, enter the sequence field name as the Key Item. If the segment has variable length, enter the name of the item containing the length as the Variable Length Item.

Use Record Definition to lay out the structure of the segment. The item structure must match the structure of the segment as DL/I presents it to the

program. Define the Key Item and Variable Length Item, if they exist, with the same length and position that they have in the DL/I segment.

If the segment is a logical child, the structure should include the concatenated key of the destination parent and the intersection data. If the segment is a concatenated segment in a logical database, the structure should include the concatenated key, the intersection data, and the destination parent segment. The sample database is a logical database. The order item segment is a concatenated segment in the database.

If a segment is redefined in the DL/I PSB using the field level sensitivity function, the structure should match the PSB view of the segment. If the segment appears in different PSBs with different item structures, use the record redefinition function to define an item structure that matches the structure in each PSB.

You must be careful to define the record structure so that its length matches or exceeds the length of the largest segment accessed. If a DL/I segment is larger than the defined record structure, the storage following the record buffer is overlaid when the segment is read. The bytes following the record buffer are checked to ensure that they are not modified when reading a database. If they are, the program ends.

Defining DL/I programs

The first step in defining a DL/I program is defining the program data structures. Defining the structures is usually done by the database administrator as described in the previous section.

The definition of maps and working storage is the same for a DL/I program as for other VisualAge Generator programs.

Defining the program logic is also very similar to the definition of the logic for a file-based program. The first step in logic definition is to define the program specifications. During program specification, enter the PSB name of the PSB part that describes the databases that the program can access.

The second step in logic definition is to define the set of functions that are run for the program. Use the same I/O options you use for a file-based program, and specify DL/I segments as I/O objects. You access segments using the following I/O options:

INQUIRY
Get a segment from a database

UPDATE
Get a segment from a database for replacement or deletion

SCAN Get the next segment of this type in a database

ADD Insert a segment in a database

REPLACE
Replace a segment in a database

DELETE
Delete a segment from a database.

You can view or change the DL/I call from the function editor. You can also request that additional DL/I functions be performed on the call.

Processing root segments

The root segment in a DL/I database is processed just like a record in an indexed file. To read a specific root segment, use an INQUIRY function and specify the segment name as the I/O object. Load the segment key field with the key of the segment you want to read before the I/O option is run.

To change a specific root segment, use an UPDATE instead of an INQUIRY function. After the segment is read, change any fields in the segment except the key and write the segment back to the database using a REPLACE function with the segment as the I/O object.

To add a specific root segment to the database, move data into the segment fields, including the key value into the key field, and use an ADD function to insert the root segment in the database.

To read the next root segment in the database, use a SCAN function with the segment as the object. If you want to start at a specific key, move the key value into the segment key item and SET record SCAN before executing the SCAN function. Note that the root segment might or might not be retrieved in key sequence order on a SCAN depending on the DL/I access method specified by the database administrator for the database. If the access method is HISAM or HIDAM, the roots are retrieved in key sequence order. If the access method is HDAM, the roots are not in key sequence order.

To delete a root segment, read the segment with an UPDATE function and follow the UPDATE with a DELETE function, naming the segment as the I/O object. If you delete a segment in a DL/I database, you delete all its dependent segments at the same time. Deleting a root deletes the root and all its dependents.

Testing the results of a DL/I call

To test the results of a DL/I call, you can use an IF, TEST, or WHILE statement in the I/O error routine to test the record state.

In addition to setting the record state after each DL/I call, VisualAge Generator also saves the status of the DL/I PCB and the CICS User Interface Block (UIB) in a set of EZE special function words. If you are experienced with DL/I calls, you can check the results of the call by using an IF or WHILE statement to check the contents of the DL/I special function words:

EZEDLDBD
Database name

EZEDLLEV
Level of lowest level segment

EZEDLSTC
Status code

EZEDLPRO
Processing options

EZEDLSEG
Name of lowest level segment

EZEDLKYL
Key length

EZEDLSSG
Number of sensitive segments

EZEDLKEY
Concatenated key

EZEDLCON
CICS condition code

EZEDLCER
CICS error code

The following is an example of how to check for a GE status code using an IF statement:

```
IF EZEDLSTC EQ 'GE';
```

VisualAge Generator distinguishes between two classes of DL/I errors. The following DL/I status codes indicate the soft errors that could be expected in a normal situation:

GA Crossed hierarchical boundary into higher level
GB End of data set, last segment reached
GD Call did not have SSAs for all levels above insert (IMS only)
GE Segment or parent segment not found
GK Different segment type at same level returned
II Segment to insert already exists in database or is non-unique

All other status codes or calls where the CICS condition code is nonzero are considered hard errors. If DL/I or CICS returns an error other than normal completion or one of the status codes listed above, program processing ends with a message describing the error. If you do not want the program to end, set either of the special function words EZEFEC or EZEDLERR to 1 in the program and include an error routine for the function. In this situation, VisualAge Generator does not issue an error message and returns to your program when hard errors are found. Your program is responsible for error handling. The program can check for errors by checking the PCB and, on a CICS system, the UIB special function words in the I/O error routine.

DL/I status codes and CICS codes are documented in the following publications:

- CICS application programmer's reference manual
- IMS or DL/I application programming manual for your system

Processing dependent segments

Processing dependent segments is similar to processing a root segment with one exception: a dependent segment is accessed through its parent chain in the database. To read a specific segment, you must set the key item of the root segment plus the key item of each segment in the dependent segment chain from the root to the I/O object segment.

For example, to read customer order 543 for customer 23 at location 12 in the customer database, you use an INQUIRY function and specify the order segment as the I/O object. Before the INQUIRY function, you move 23 into the key item for the customer segment, 12 into the key item for the location segment, and 543 into the key item for the order segment. The INQUIRY function searches for customer order 543 for customer 23 at location 12 and reads the order if it was in the database.

You can change or delete a specific order by using an UPDATE function to read the segment. Initialize the keys for an UPDATE I/O option the same as you initialize the keys for an INQUIRY I/O option. Once you have read the order segment, a REPLACE or DELETE I/O option that specifies the segment as the I/O object replaces or deletes the segment in the database.

To add a new order for customer 23 at location 12, use an ADD function. Before the ADD function, set the items in the order segment including the order number. Also, set the customer number (the root segment key) to 23 and the location number (the location segment key) to 12. This gives DL/I the information required to correctly position the new order in the database.

To read the next order segment in the database, use a SCAN option specifying the order segment as the I/O object.

SCAN function variations

Once you view the DL/I call that is generated for an I/O option you can do any of the following:

- Replace or delete segments
- Scan within a parent
- Set a SCAN position
- Search on partial keys
- Determine the parents of a segment retrieved on a SCAN

Replacing or deleting segments

If you want to replace or delete segments that are retrieved with a SCAN I/O option, edit the SCAN function and edit the DL/I call. Specify **Scan for Update**. You can then follow the SCAN function with a REPLACE or DELETE function for the same segment.

Scanning within a parent

The default SCAN function starts at the current position in the database and searches through the entire database for the next occurrence of the object segment under any parent. To limit the scope of the SCAN to the currently established dependent chain, edit the DL/I call for the SCAN function and select the SCAN in parent option. An NRF condition is returned on the SCAN following the reading of the last segment under the current parent. NRF is also returned if there are no segments of the requested type in the current parent chain.

To process all the segments of the same type under a specific parent, use an INQUIRY function to read the parent segment and to set the database position pointer. To read the dependents in one at a time, repeat a SCAN function with SCAN in parent specified. End the loop on the NRF condition.

In the sample customer database, if you want to process all the orders for one specific customer location, use an INQUIRY function to read in the required location segment. Then use a SCAN function to read in the order segments one at a time. If you specify SCAN in parent, the NRF condition is returned when there are no more orders for the location.

Setting SCAN position

Before beginning a SCAN loop, your program must establish position in the database. You do this in one of the following ways:

- Use an INQUIRY or UPDATE I/O option to establish position. The example in the previous section used INQUIRY to set the position pointer to a specific location segment so that the SCAN in parent specification can be used to read each order for that location.
- Set the key fields of the object segment and all segments of the dependent chain from the root to the object segment the same as you set the key fields for the object segment and all the segments in the dependent chain from

the root to the object for an INQUIRY I/O option. Define the SET record SCAN statement instruction before the SCAN function. VisualAge Generator creates a DL/I call that acts as if the position pointer were set to point just before the position indicated by the key fields.

The set scan indicator can be used only if you have not modified the SSA list for the SCAN and have not specified SCAN in parent. The running program is ended if the set scan indicator is on for a SCAN function for a modified SSA list. The indicator is reset on the first I/O function for that segment that follows the setting of the indicator.

Searching on partial keys

If you know only the first part of a segment key, you can retrieve the first segment with a key that begins with that partial key. Move the partial key into the key item; set the keys of the other segments in the path from the root to the object segment, and use the SET record SCAN function as described under "Setting SCAN position" on page 97

Determining the parents of a segment retrieved on a SCAN

The EZEDLKEY special function word contains the concatenated segment key of the lowest level segment retrieved on any DL/I call after the call completes successfully. The concatenated key consists of the contents of the sequence fields of all the segments in the path from the root to the object segment concatenated in a single string.

In the sample database, if you retrieved an order segment using SCAN, EZEDLKEY contains a 24-byte string consisting of the customer number followed by the location number, followed by the order date and number. To access the individual parts of the key, you define a 24-character item in working storage and subdivide it into the 6-byte customer number, the 6-byte location number, and the 12-byte order date/number. In the statements following the SCAN, you could move EZEDLKEY to the working storage item and reference the individual segment keys in working storage.

Additional function through SSA list modification

You can use additional DL/I functions by modifying the SSA list that is built by VisualAge Generator for the I/O option. Refer to the VisualAge Generator Developer online help system for more information.

Using path calls to access multiple segments at the same time

If your I/O object is a dependent segment, you can read in any of the segments on the path from the root to the object with the same call that you retrieve the object. To do this, edit the DL/I call for the function and enter a D command code for each segment you want to retrieve on the call. You can modify the call by using D command codes to read in the customer segment,

the customer location segment and the order segment on the same call. To determine if the call completed successfully, use the I/O error routine to test the state of the object segment.

If you use the D command code on an UPDATE or SCAN for UPDATE function, the subsequent REPLACE function replaces each segment retrieved. You can selectively prevent replacement of one or more segments by naming the selected segments and specifying an N command code in the SSAs for the REPLACE function.

The default DL/I call built for a DELETE function that follows an UPDATE or Scan for Update with D command codes does not delete each segment retrieved. It deletes only the I/O object segment.

Using non-key fields as search arguments

You can use any field in a segment as a search argument on a DL/I call by modifying the SSA list for the call.

For example, if you wanted to scan through the customer database and retrieve the customer segment and status segment for each customer with a credit balance greater than a specified amount, you define the DL/I call search arguments as follows:

1. You want to search on the credit balance field (STFCSBL) in the status segment. To do this, define an item in working storage (CHECKBAL) that contains the specified amount that you want to search for. Make CHECKBAL a 12-byte numeric field just like STFCSBL.
2. Use a SCAN option with the status segment (STSCSTA) as the I/O object.
3. Edit the DL/I Call for the function and add a qualification statement to the SSA that retrieves the next segment found that the contents of the credit balance field (STFCSBL) are greater than the amount in CHECKBAL.
4. Add another SSA in the list with a D command code that retrieves the customer record with the status segment.

Scanning through all database segments with a single function

You can use a single function to scan all segments in a database. If a DL/I get next (SCAN) call is issued with no SSAs, DL/I returns the next segment in the database regardless of its type. You can use this type of call with any of the following techniques:

1. Define a SCAN function for the largest segment in the database.
2. Edit the default DL/I call for the function and delete the single SSA in the default call.
3. Define a set of records with structures matching the other segments in the database. Define them as redefined records for the I/O object record.

4. Test the EZEDLSEG special function word after the SCAN to determine which segment was retrieved on the SCAN.
5. Access the segment that is retrieved from the redefined record structure or move the data from the redefined record to the actual segment of the record that was retrieved. VisualAge Generator Server for MVS, VSE, and VM does not automatically recognize that an alternate segment has been read and move the segment data for you.
6. If the largest segment in the database is a variable-length segment, define the segment to VisualAge Generator as fixed length (the length should be equal to the length of the largest possible returned segment). The 2-byte length code should still be defined at the beginning, so you can check the actual length returned by DL/I.

Accessing the same segment in two data structures

Sometimes the same segment appears in two or more PCB entries in the same PSB. This can happen when a structure is accessed by a secondary index as well as the primary path or when the same structure is repeated in more than one PCB in order to maintain multiple positions in the database.

VisualAge Generator Developer creates default DL/I calls based on the first PCB that the object segment appears in. If you want to access the object segment by using a later PCB in the PSB, edit the DL/I call and select the number of the other PCB you want using the database identifier field on the DL/I Call Definition window.

Sharing a function when programs use different PSBs

You can use the same DL/I function in programs that use different PSBs as long as the PCB associated with the function is included in both PSBs. The PCB does not have to be at the same offset in each PSB, but does have to have the same database name and segment structure.

Use a technique explained in this section when you want to reuse a function with a database that is used in more than one PCB in each PSB. Consider two IMS PSBs that have the structure shown in Figure 8 on page 101:

APPL1				APPL2			
PCB Type	Data-base Name	Segment Name	Parent Segment Name	PCB Type	Data-base Name	Segment Name	Parent Segment Name
TP				TP			
TP				TP			
DB	DB00	SEG00		DB	DB01	SEG01	
DB	DB01	SEG01				SEG02	SEG01
		SEG02	SEG01			SEG03	SEG02
		SEG03	SEG02			SEG04	SEG02
		SEG04	SEG02	DB	DB01	SEG01	
DB	DB01	SEG01				SEG02	SEG01
		SEG02	SEG01			SEG04	SEG02
		SEG04	SEG02				

Figure 8. Sample structure for an IMS PSB

APPL1 has the same physical database (DB01) used for two different PCBs. This represents the need to have two views of this database, both having similar segments (SEG01, SEG02, and SEG04). APPL2 has a similar PCBs, but they are at different offsets because database DB00 is not included in the APPL2 PSB.

The database name in the VisualAge Generator PSB is never used to determine the physical database. It is only used in determining which PCB to use in the DL/I call. Therefore, to be able to easily refer to the different PCBs for DB01, the VisualAge Generator PSBs can use a “dummy” database name for the second occurrence. Note that the only change between Figure 8 and Figure 9 on page 102 is that the second occurrence of DB01 in both PCBs has been renamed DB01X.

APPL1P				APPL2P			
PCB Type	Data-base Name	Segment Name	Parent Segment Name	PCB Type	Data-base Name	Segment Name	Parent Segment Name
TP				TP			
TP				TP			
DB	DB00	SEG00		DB	DB01	SEG01	
DB	DB01	SEG01				SEG02	SEG01
		SEG02	SEG01			SEG03	SEG02
		SEG03	SEG02			SEG04	SEG02
		SEG04	SEG02	DB	DB01X	SEG01	
DB	DB01X	SEG01				SEG02	SEG01
		SEG02	SEG01			SEG04	SEG02
		SEG04	SEG02				

Figure 9. Example where DB01 in both PCBs has been renamed DB01X.

Two functions can be defined:

- PROC-INQUIRY-A to do an INQUIRY on SEG04 using DB01
- PROC-INQUIRY-B to do an INQUIRY on SEG04 using DB01X

These two functions can then be used in programs APPL1 (using PSB APPL1P and IMS PSB APPL1) and APPL2 (using PSB APPL2P and IMS PSB APPL2) without requiring further modifications and without requiring the database identifier to be reset when switching between the two programs.

Using a secondary index

Sometimes the DL/I PSB indicates that a database structure is to be accessed with a secondary index. When this situation occurs, your database administrator had entered an index key name next to the root segment in the PSB definition. VisualAge Generator uses the index key instead of the segment key item as the search field in creating DL/I calls that reference that segment. If the field is not defined in the segment itself, you must define the field as an item in working storage with the correct type and length. When you set up an INQUIRY function that references the segment, set the index key instead of the segment key item before running the I/O option.

In CICS for MVS/ESA, your program ends abnormally with an `ADLA CICS ABEND` if you have defined the secondary index with duplicate key values not allowed and your program attempts to insert a duplicate key in the secondary index. (On CICS for VSE/ESA systems, a NI status code is returned). Therefore, on CICS for MVS/ESA systems, test for a duplicate key already in the database before doing the insert.

Using CSPTDLI service routine for database calls

Some DL/I functions cannot be implemented using the DL/I Call Definition function of function definition. The CSPTDLI service routine can be used to implement these DL/I calls. For example, there is not an I/O option to generate a FLD (field) call for a main storage database (MSDB). However, you can use the FLD call by defining a CALL CSPTDLI statement with the same parameters you use if you were coding in COBOL. The POS (position) call for data entry databases (DEDBs) can also be defined with the CSPTDLI service routine.

In CICS, CSPTDLI should not be used for PSB scheduling. The program automatically schedules the PSB as required.

In IMS, CSPTDLI should not be used for get unique, insert, rollback, or checkpoint calls to the I/O PCB in a main transaction program or in a batch program that is called by a transaction program.

The EZEDL status words are not set on a CSPTDLI call. If you must check the DL/I status code after using CSPTDLI, you must move the EZEDLPCB(n) special function word (where n is the PCB number that you used in the CSPTDLI call) to a working storage record and then examine the status area of the working storage record. Refer to your system's IMS or DL/I program programming manual for more information about the syntax and for examples of these calls.

Sharing a PSB with a called program

Called and calling programs cannot both be DL/I programs unless they share the same program PSB or unless a commit is done to end the PSB before each call or return to a program that uses a different PSB.

If the called program uses DL/I, it can do any of the following:

- Specify a PSB and choose not to include EZEDLPSB or EZEDLPCB in its parameter list (CICS only).
- Include EZEDLPSB in the program's parameter list to receive a complete PSB from the calling program.
- Include one or more EZEDLPCB(n) in the program's parameter list to receive specific PCBs from the calling program.

Each VisualAge Generator program maintains its own copy of a record. You cannot read a segment from a database in one program and reference the items in that segment in another program unless the record was explicitly passed as a parameter from one program to the other.

REPLACE and DELETE functions must be included in the same program that read the segment for update in the first place. If a program is invoked through a non-VisualAge Generator program, an UPDATE or Scan for Update and a subsequent REPLACE or DELETE must be done within the same call to the VisualAge Generator program.

Passing EZEDLPSB

The EZEDLPSB special function word is a structure containing a PSB name and a pointer to the CICS User Interface Block (UIB). The UIB is the standard way of getting addressability to database program control blocks (PCBs) in the CICS for MVS/ESA or CICS for VSE/ESA environment. The UIB is simulated in the non-CICS environments for portability and must point to the PCB address list passed to the initial program when the IMS transaction, MVS/TSO program, or batch job step was started.

As shown in Figure 10, the 12-byte area passed by the VisualAge Generator program contains the PSB name and a 4-byte address (simulated UIB) that addresses a pointer to the PCB list to be used by the called program. This pointer to the PCB list is saved by programs started by the IMS control program.

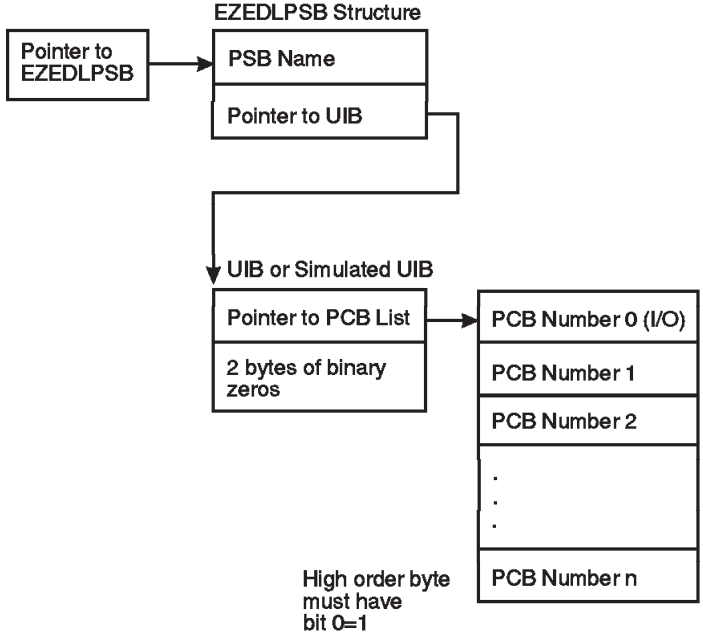

Figure 10. Structure of EZEDLPSB

Note: If you are using the COMMDATA parameter format for CICS, refer to *VisualAge Generator Client/Server Communications Guide* for information on how EZEDLPSB is passed.

If a non-VisualAge Generator program is passing control to a VisualAge Generator program, the program must pass the 12-byte area described above, with a pointer to a pointer that addresses the PCB list in bytes 9–12 (simulated UIB). This pointer must address an assembler format PCB list (that is, it must point to an area containing the PCB pointers in contiguous storage).

For special considerations in testing DL/I programs, see the DL/I testing considerations section.

Passing EZEDLPCB

Individual PCBs can be passed on a CALL statement by subscripting EZEDLPCB with the PCB number to be passed. The passed PCBs are associated to the PCBs defined by EZEDLPCB in the parameter list of the called program. Passing PCBs as parameters is the standard method of getting addressability to PCBs in non-CICS environments and is useful in CICS environments for calling programs that access a database shared by multiple transactions. Each transaction can have a different PSB, that contains the PCB for the shared database at a different offset in the PSB.

The subscript for the EZEDLPCB special function word must be a numeric literal. If the EZEDLPCB special function word is not subscripted on the CALL statement or in the parameter list, it defaults to the first PCB.

Only PCBs passed on the call can be used by the called program. If the PSB was not scheduled prior to the CALL statement, the PSB is scheduled automatically before the CALL is performed.

A non-VisualAge Generator program calling a generated program can pass a PCB to the generated program as follows: If a high-level language call is used, specify the PCB as a standard argument on the call passing the argument by reference. If a CICS LINK is used, pass the PCB pointer in the COMMAREA. The PSB must be scheduled prior to calling or linking to the generated program. Refer to *VisualAge Generator Client/Server Communications Guide* for more information on how parameters are passed in CICS environments.

You cannot specify EZEDLPSB and EZEDLPCB on the same CALL statement.

For special considerations in testing DL/I programs, see the DL/I testing considerations section.

Sharing a PSB with a transferred-to program using XCTL

Transfers from a program to a main transaction or main batch program (or from a program to a program) in the MVS/TSO, MVS batch, and IMS BMP environments can be implemented using an OS XCTL to the generated program. You can pass the special function word EZEDLPSB or a list of PCBs as parameters on the OS XCTL macro.

Passing the EZEDLPSB special function word

One or two parameters are passed to the receiving program on the OS XCTL macro. The first parameter is the working storage buffer consisting of a 2-byte length field, an 8-byte filler field, and the working storage data. The second parameter, used only if the generated program is a DL/I program, is the EZEDLPSB special function word. Figure 11 shows the format of the parameter list that the transferring program should pass. See "Sharing a PSB with a called program" on page 103 for the format of the EZEDLPSB special function word.

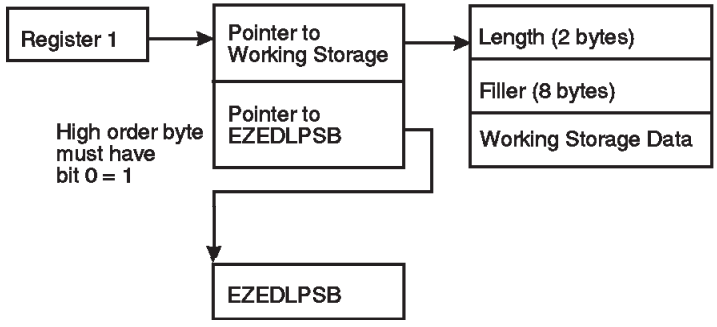

Figure 11. The format of the parameter list that the transferring program should pass

Passing a list of PCBs

If the generated program is a DL/I program, a program can transfer to the generated program, passing a list of PCBs as parameters on the OS XCTL macro. The generated program receives control in this way if it is started as a DL/I MVS batch job. The PCBs must match the PSB definition expected by the generated program.

The working storage record is initialized to blanks if the generated program is started with this interface.

Sharing a PSB across environments

If you are defining a PSB for a program that you also plan to generate for CICS for MVS/ESA, and if TP (including the I/O PCB) and GSAM PCBs are included in the PSB definition, the same IMS PSB definition can be shared between CICS for MVS/ESA, IMS BMP, IMS/VS, MVS/TSO, and MVS batch environments. The PSB must be generated with CMPAT=YES on the PSBGEN macro. The TP PCBs are ignored in CICS for MVS/ESA and cannot be referenced using EZEDLPCB or CSPTDLI calls in the CICS environment. The PSB must comply with the compatibility considerations for IMS/VS. For CICS for MVS/ESA, the actual PSB name is moved into EZEDLPSB prior to the first use of a DL/I segment.

Sharing the IMS PSB definition with CICS for MVS/ESA is not recommended if the DL/I work database is used. The ELAWORK PCB in the PSB gets a PSB work area of approximately 64K bytes each time the PSB is scheduled. The work area is obtained to support a database (the DL/I work database) that is not accessed in the CICS for MVS/ESA environment. To avoid this problem, use an alternate PSB for the CICS for MVS/ESA environment with an alternate PCB at the same position as ELAWORK. Associate this PCB with only the root segment of the ELAWORK database or with any other small database segment. The PCB is acting as a place holder and is not accessed by the generated program in the CICS for MVS/ESA environment.

Note: DL/I PSBs for VSE environments cannot include TP PCBs nor GSAM PCBs; therefore, you cannot share the DL/I PSB definition with the IMS environments. The VisualAge Generator PSB definition can include the TP and GSAM PCBs for the VSE environments. Any TP or GSAM PCBs are ignored (eliminated) by the VisualAge Generator Developer when generating for the VSE environments.

Assuring data integrity between CONVERSE I/O options

If your program accesses a database that is to be shared by multiple users, you should define your program to run in segmented mode or you should define your program to set the EZECNVCM special function word to 1. In both cases, the program commits database changes at each CONVERSE I/O option. This ensures that your program does not lock out any database records for long periods of time while waiting for a response from a program user. However, you must define programs to prevent two program users from changing the same record at the same time, or to reestablish the position in the database if your program is scanning the database.

Note: IMS always commits database changes at each CONVERSE function; in CICS, the PSB also ends when updates are committed.

You should use a compare and update technique to ensure that another program has not modified the record while it was being displayed to the program user if the following is true:

- Changes are committed at each CONVERSE, and
- Your program reads a segment, displays the segment contents on a map, reads changes from the map, and then writes the changes back to the database

Use the following steps to compare and update the record:

1. Read the segment.
2. Save a copy of the segment in working storage.
3. Display the segment contents to the program user.
4. Read updates from the terminal.
5. Read the segment with an update function.
6. Compare the segment just read with the saved segment.
7. If not equal, go back to step 2.
8. If equal, update the segment from the terminal input.
9. Write the segment back to the database with a replace function.
10. Display a message confirming that the segment has been updated.

An alternative to saving and comparing large segments is to store the current date and time as part of the segment each time the segment is modified. The program can then determine whether a record has been updated by saving and comparing the time stamp instead of the entire segment.

DL/I considerations for the CICS environment

In the CICS environment, PSB scheduling is handled differently than in non-CICS environments. The following sections describe PSB scheduling, how to use an alternate PSB at run time and how to share a PSB with a called program.

Understanding PSB scheduling

A CICS DL/I program must schedule a PSB before it can access databases defined in that PSB. The program must end the PSB when database access is complete. VisualAge Generator Server for MVS, VSE, and VM automatically handles PSB scheduling for a program. However, you need to know when the PSB is scheduled because the following functions are affected by PSB scheduling:

Segment record locking
: Segment update locks are released when the PSB is ended

Database positioning
: Database position is lost when the PSB is ended

Update commitment
Changes are committed (written to the database) when the PSB is ended

The PSB is scheduled whenever a DL/I call is issued for the program and the PSB is not currently scheduled. The PSB named in EZEDLPSB is the PSB scheduled.

When using a CALL or DXFR statement to transfer program control, the transferred-to program does not have to use the same PSB the transferring program uses.

The PSB ends whenever a CICS SYNCPOINT or SYNCPOINT ROLLBACK is issued. SYNCPOINTs occur when one of the following occurs:

- The top-level program in a run unit ends successfully and returns control to CICS.

  For CICS, a run unit is equivalent to a single transaction and consists of all VisualAge Generator programs and non-VisualAge Generator programs that transfer control among themselves using a DXFR or CALL statement. For non-VisualAge Generator programs, this also includes any transfer that uses a CALL statement, CICS LINK command, or CICS XCTL command.
- A program uses a CONVERSE I/O option, and any of the following is set to 1:
  - EZESEGM special function word (segmented mode)

    The EZESEGM special function word defaults to 1 if the program is defined as segmented.
  - EZECNVCM special function word (CONVERSE commit)
  - EZEDLTRM special function word (end the PSB at CONVERSE) if the program uses DL/I

    The best time for a commit point to occur is after terminal output and before the next terminal input. A commit point at terminal I/O synchronizes updates to the database and confirmation messages to the program user.
- A transfer using an XFER statement occurs.
- A program calls either the EZECOMIT or COMMIT service.
- A transfer using a DXFR statement occurs, a PSB is scheduled, and one of the following occurred:
  - Transfer to a non-VisualAge Generator program and a PSB is scheduled.
  - The /SYNCDXFR generation option was specified for the transferred-from program

- The /NOSYNCDXFR generation option was specified for the transferred-from program and different PSB names were identified in the program specifications for the two programs.

- A VisualAge Generator called DL/I program returns to the calling non-VisualAge Generator program, the PSB was not passed using the EZEDLPSB special function word, and PCBs were not passed using the EZEDLPCB special function word.

A SYNCPOINT ROLLBACK occurs when:

- A VisualAge Generator program calls the EZEROLLB or RESET service.
- A program ends because of an error condition.

When a rollback occurs, all changes that were made to databases and recoverable files since the start of the LUW are backed out.

Using an alternate PSB at run time

VisualAge Generator uses the PSB named in the program specification to create DL/I calls for the segments defined in the PSB and to validate any changes you make to the DL/I calls. In the CICS environment, VisualAge Generator Server for MVS, VSE, and VM also uses the PSB name for PSB scheduling at the time the program is run.

Sometimes, however, you might want to use an alternate PSB with the same program. The alternate PSB describes databases with the exact same structure as the program PSB, but the databases themselves might be different. For example, you could have a set of test databases for program development and a corresponding set of production databases that contain the real data for production.

If you want to use an alternate PSB for any reason, your program can dynamically change the PSB that is scheduled by moving the alternate PSB name into the special function word EZEDLPSB before running the first DL/I function. The alternate PSB must match the program PSB except that the database names can be different.

Sharing a scheduled PSB with a called program

Called and calling programs cannot both be DL/I programs unless they share the same program PSB or unless a commit is done to end the PSB prior to each call or return to a program that uses a different PSB. The PSB is shared by specifying the special function word EZEDLPSB or EZEDLPCB as a parameter passed on the call for both the called and the calling program.

If the called and calling programs are both VisualAge Generator programs, and if the PSB was scheduled before the call, VisualAge Generator does not reschedule the PSB in the called program.

You can share a PSB between VisualAge Generator programs and non-VisualAge Generator programs. When EZEDLPSB is passed as a parameter, a 12-byte area is actually passed. The first 8 bytes contain the PSB name; the final 4 bytes contain the address of the CICS User Interface Block (UIB). If the PSB is not scheduled, the UIB address is 0.

When sharing a PSB between VisualAge Generator programs and non-VisualAge Generator programs, the called program should check the UIB address before scheduling. If the UIB address is not 0, the PSB should not be rescheduled. If the called program ends the PSB, it must set the UIB address field to 0. If the PSB is scheduled again, the UIB address field should be set to the UIB address returned by CICS.

If a program needs to share a scheduled PSB with a called VisualAge Generator program, it should pass a 12-byte area to the program. Again, the first 8 bytes should contain the PSB name and the next 4 bytes should contain the UIB address. If the PSB is not scheduled, the UIB address should be 0. On return, the UIB address reflects the current scheduling status of the PSB.

Recovering after a deadlock in record queuing

Updated records are not actually written to the database until the PSB is ended. Your program obtains exclusive use of these records until PSB termination because the UPDATE function locks out other programs from changing the record again until it is actually written to the database. This can result in a deadlock situation where your program has some record locked and now wants to change records locked by another program that in turn needs the records you have locked. When CICS detects a deadlock situation, it abnormally ends one of the programs, backs out the changes it has made to the database, and writes an error message to the terminal explaining why the program ended.

If having a program abnormally end is unacceptable to the users of the program, you can define your program as *restartable* in the CICS tables. See "Restarting VisualAge Generator programs after a DL/I deadlock" on page 112 for information on restarting programs.

If the program is restartable and CICS detects a deadlock situation, CICS backs out the changes that the program has made since the PSB was scheduled and restarts the program from the beginning of the transaction.

If your program is running in segmented (pseudoconversational) mode, the most recent segment of the program is restarted, and you do not need any special restart code in the program.

If a conversational program is restarted, the program is entered at the top of the program. You can determine that the program was restarted by having the

program test for a value of 1 in the EZEDLRST special function word. If EZEDLRST is 1, you can write a message to the program user explaining that the program was restarted because of a deadlock and then display the initial program map again.

Restarting VisualAge Generator programs after a DL/I deadlock

When the DL/I program isolation facility is used, deadlocks can occur between two transactions locking on the same record. The programs try to update the same record at the same time. If both updates are accepted, one of the changes is lost. If CICS for MVS/ESA or CICS for VSE/ESA detects that data is lost, it abnormally ends the transaction with an `ADLD` abend code.

The VisualAge Generator Server for MVS, VSE, and VM abend handler requests restart for programs that end with a deadlock abend. CICS for MVS/ESA or CICS for VSE/ESA restarts the transaction from the beginning if:

- You have specified DTB=YES and RESTART=YES in the PCT for the transaction program.
- The CICS for MVS/ESA or CICS for VSE/ESA transaction restart program (DFHRTY) specifies to restart the transaction.
- The temporary storage queues are defined as recoverable.

Otherwise, CICS for MVS/ESA or CICS for VSE/ESA writes a message indicating the reason for program termination.

A restarted program is a program that is started again from the beginning of the last transaction that was running. All changes to databases that were made since the program PSB was last scheduled are rolled back.

The distributed version of the CICS for MVS/ESA or CICS for VSE/ESA restart program (DFHRTY) does not restart VisualAge Generator transactions. You can add code to DFHRTY to restart VisualAge Generator transactions. Your program should check that the following is true:

- The current abend code is `ADLD`.
- The transaction identifier is the identifier of the transaction you want restarted.
- The restart count is less than the specified number. This is a restart loop check.

If all checks are met, your program should set the restart flag on, indicating to CICS for MVS/ESA or CICS for VSE/ESA that restart is to continue. For more information on modifying DFHRTY, refer to the recovery, restart and customization manuals for your CICS for MVS/ESA or CICS for VSE/ESA system.

Programs running in segmented mode must have all the CICS for MVS/ESA or CICS for VSE/ESA resources (that are to be updated) defined as recoverable if restart is requested. The names of the temporary storage queues that are created by the segmentation function of VisualAge Generator Server for MVS, VSE, and VM are a combination of the user's terminal identifier appended to a 4-character prefix. The prefixes used are in the form of X'EE' followed by either WRK or MSG.

You should not request restart for conversational programs unless you have designed the program to handle restart. A program can test the EZEDLRST special function word to see if the current program transaction has been restarted. One simple way of designing a program for restart is to have the program test EZEDLRST whenever it begins. If the restart flag is on, a special message or map should appear to user 1 explaining that the transaction was restarted at the beginning because user 2 was changing the database at the same time. User 1's changes were backed out, and the program was restarted to prevent the changes from being lost.

Accessing distributed DL/I databases

Programs running on CICS for MVS/ESA, CICS for VSE/ESA, or CICS for OS/2 systems can access DL/I databases on remote systems by calling an CICS for MVS/ESA or CICS for VSE/ESA called batch server program that runs on the system where the database is located.

DL/I considerations for non-CICS environments

In non-CICS environments, PSB scheduling is handled differently than in CICS environments. The following sections describe PSB scheduling and the use of an alternate PSB at run time.

Understanding PSB scheduling

During program execution for MVS/TSO, IMS/VS, IMS BMP, MVS batch, and VSE batch processing, DL/I initialization schedules a single PSB. This PSB is the only one available for one MVS/TSO invocation, batch job step, or IMS transaction.

All programs and non-VisualAge Generator programs in the run unit must share the same PSB. The run unit includes all programs that are called or transferred-to using a DXFR statement. For MVS/TSO, IMS BMP, MVS batch, and VSE batch, the run unit also includes all programs that are transferred-to using an XFER statement. For IMS/VS, the run unit for a program transferred-to using an XFER statement differs from that of the transferring program.

For MVS/TSO, you specify the name of the PSB to be used in the runtime CLIST used to run the program. The PSB is scheduled when you run the runtime CLIST for the program.

For IMS/VS, you specify the PSB that is to be used in the IMS system definition. The IMS PSB must have the same name as the program name. The PSB is scheduled at the beginning of the IMS transaction.

For IMS BMP, MVS batch, and VSE batch, you specify the name of the PSB to be used in the JCL used to run the batch job. The PSB is scheduled at the beginning of the IMS BMP, MVS batch, or VSE batch job.

Understanding commit points and the logical unit of work

A logical unit of work (LUW) ends whenever a commit point or a rollback occurs.

A commit point occurs when the following happens:

- The top-level program in a run unit ends successfully.

  For MVS/TSO, MVS batch, VSE batch, and IMS BMP, a run unit consists of all VisualAge Generator programs and non-VisualAge Generator programs that transfer control among themselves using an XFER, DXFR, or CALL statement. For non-VisualAge Generator programs, this also includes any transfer that uses an OS XCTL macro or a CALL statement.

  For IMS/VS, a run unit is equivalent to a single transaction and consists of all VisualAge Generator programs and non-VisualAge Generator programs that transfer control among themselves using a DXFR or CALL statement. For non-VisualAge Generator programs, this also includes any transfer that uses a CALL statement.
- A program uses a CONVERSE I/O option and any of the following is set to 1:
  - EZESEGM special function word (segmented mode). The EZESEGM special function word defaults to 1 if the program is defined as segmented.
  - EZECNVCM special function word (CONVERSE commit)
  - EZEDLTRM special function word (end the PSB at CONVERSE) if the program uses DL/I.

  The best time for a commit point to occur is after terminal output and before the next terminal input. A commit point at terminal I/O synchronizes updates to the database and confirmation messages to the program user.
- For MVS/TSO, when a transfer using an XFER statement occurs for a segmented or single segment program.

- For MVS/TSO, MVS batch, and batch-oriented IMS BMP programs, a program transfers using an XFER statement and the /SYNCXFER generation option is specified for the transferred-from program.
- A program calls the EZECOMIT or COMMIT service.

  For MVS/TSO, MVS batch, or VSE batch, VisualAge Generator programs that do not use DL/I issue a commit point only if the program has made changes to an SQL table. A commit point does not occur for changes to an SQL table made by a non-VisualAge Generator program.

  For IMS/VS and transaction-oriented IMS BMP programs (programs that scan a serial file associated with the I/O PCB), EZECOMIT is ignored. A commit point occurs whenever there is a get unique to the I/O PCB.
- For IMS/VS and transaction-oriented IMS BMP programs, a program does a successful get unique to the I/O PCB.

A rollback occurs when the following happens:
- A VisualAge Generator program calls the EZEROLLB or RESET service
- A program ends because of an error condition

When a rollback occurs, all changes that were made to databases and recoverable files since the start of the LUW are backed out. Rollback does not affect DL/I databases in the VSE batch environment.

Using an alternate PSB at run time

The actual DL/I PSB name you use at run time might differ from the name you specified as the PSB name during program specification. However, for IMS/VS, IMS BMP, MVS batch, and MVS/TSO, the PCB number, type, and order must match in the IMS PSB and the VisualAge Generator PSB. Although the database structures must match, the database names do not have to match the name specified in VisualAge Generator PSB definition.

Using symbolic checkpoint and restart functions (MVS Batch and IMS BMP Only)

When you run a batch program, you can use EZECOMIT to periodically commit database updates. Alternatively, you can use CSPTDLI to implement symbolic checkpoint and restart functions.

Both EZECOMIT and the symbolic checkpoint function commit database updates. However, the symbolic checkpoint function also enables you to save information, such as control totals or the key of the last database record that was processed when a commit point occurred. If the program does not complete successfully, it is backed out to the last commit point. If you have saved information using symbolic checkpoint, when you restart the program, you can restore the saved data using the restart (XRST) call. You can use this information to resume processing at the point in the database where

processing stopped. Refer to the IMS application programmer's manual for more information about the symbolic checkpoint and restart functions.

DL/I Considerations for the Test Facility

This section gives information on using DL/I in the Test Facility. Topics are as follows:

Note: You cannot test these functions in the test facility:

- Calls to the I/O or TP PCBs using CSPTDLI.
- I/O or TP PCB references using EZEDLPCB.
- CICS deadlock restart using EZEDLRST.

Also, if your program accesses message queues as serial files, you need to test with serial files instead of using I/O or TP PCB DL/I calls.

Setting up the Test Facility for DL/I

To provide access to DL/I databases at test time, do the following:

1. In the case of VisualAge for Java, go to the VAGen Parts Browser menu bar and click **Windows→Options**. The VisualAge Generator Options screen appears.

   In the case of VisualAge Smalltalk, go to the VisualAge Organizer window menu bar and click **Options→VAGen Preferences**. The VisualAge Generator Preferences screen appears.
2. In the left pane, click **DL/I**.

You select options in each of the following categories:

- **Default DL/I PSB name**
- **Target environment**
- **DL/I Database Access Middleware**
- **Options**

Default DL/I PSB name

In relation to **Default DL/I PSB name**, you specify how the test facility identifies which IMS PSB to schedule. Three options are available:

- If you select **Main program name**, the PSB scheduled has the same name as the program, as is necessary for IMS/VS transactions.

- If you select **EZEDLPSB**, the PSB scheduled has the name equal to the setting of that EZE word; for details, see the EZEDLPSB entry in the *Programmer's Reference*.
- If you select **Prompt for PSB name**, the test facility asks you to specify a PSB name during each test.

Selecting a different way to specify the PSB name affects the next use of that information; you do not need to wait for a new invocation of the test facility.

Target environment

In relation to **Target environment**, you specify the environment that the test facility is to simulate when performing commits and rollbacks and when scheduling, sharing, and terminating PSBs. The options are as follows:

- IMS BMP
- IMS VS
- MVS Batch
- MVS CICS
- MVS TSO
- VSE Batch
- VSE CICS

For details on how the target environment affects your program's interaction with DL/I databases, see "DL/I considerations for the CICS environment" on page 108, "DL/I considerations for non-CICS environments" on page 113, and "Understanding how the test facility handles commits and rollbacks" on page 119.

Note: At test time, if the target environment is MVS CICS or VSE CICS and if the program is handling hard errors (EZEFEC = 1) and PSB scheduling fails, the test facility emulates the target environment by setting EZEDLCON and EZEDLCER return fields with the appropriate CICS return codes.

Selecting a different target environment has an effect only when you next invoke the test facility.

Although you specify an environment that is simulated, the program at test time executes in one of a smaller set of environments, as described in relation to Database Access Middleware.

DL/I Database Access Middleware

In relation to **DL/I Database Access Middleware**, you specify the product used by the test facility to access DL/I. The available options depend on the type of workstation you use.

The options on workstations that use Windows 2000 or Windows NT:

- Micro Focus Mainframe Express
- VisualAge Remote MVS
- VisualAge Remote VSE

The options on workstations that use OS/2:

- Micro Focus
- VisualAge Remote MVS

If you select **Micro Focus** or **Micro Focus Mainframe Express**, your purchase and configuration of products sold by MERANT determines whether the test facility accesses DL/I remotely on MVS or in a workstation-based simulation. If you select one of the VisualAge-related options instead, a built-in facility (if installed) provides remote DL/I access:

- In the case of **VisualAge Remote MVS**, VisualAge brings up an IMS batch environment on MVS, and you are able to access DL/I in the test facility even if your organization lacks an IMS Transaction Manager on MVS.
- In the case of **VisualAge Remote VSE**, the DL/I requests are executed on VSE under the control of CICS for VSE/ESA; and when you specify the target environment to be simulated in test facility, you must select either VSE Batch or VSE CICS.

Neither of the two VisualAge options supports a local DL/I simulation or provides access to IMS Fast Path databases.

The Micro Focus support of DL/I on MVS gives access to IMS Fast Path databases, but requires that the IMS Transaction Manager be present for any DL/I access on MVS. In relation to Windows 2000 or Windows NT, the MERANT products support both remote DL/I access and a local simulation. In relation to OS/2, however, the newest versions of the MERANT products support neither remote DL/I access nor a local simulation, although those capabilities are available if you use older versions of those products.

For details on the prerequisites for each middleware option, access the appropriate web page from the list displayed under *Additional Links* at the following web site:

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

After you have gone to the Hardware and Software Prerequisites web page, review the section on *Hierarchical database access*.

Selecting a different middleware option has an effect only when you next invoke the test facility.

Options

Selecting the **SYNCXFER** or **SYNCDXFR** option causes the test facility to fulfill the behavior provided by a generation option of the same name. For details, see the *VisualAge Generator Generation Guide*.

During a test, selecting or clearing the **SYNCXFER** or **SYNCDXFR** option affects the next use of that information; you do not need to wait for a new invocation of the test facility.

Understanding how the test facility handles commits and rollbacks

In relation to commits and rollbacks, the test facility does the following in most cases:

- Performs a commit or rollback in response to an EZECOMIT or EZEROLLB in your code, if the behavior would occur at runtime.
- Performs a commit when a test ends normally.
- Performs a rollback when a test ends abnormally either because a hard error occurs and your code does not handle the error (EZEFEC=0) or because you end a test before the last statement is executed.

Exceptions are as follows:

- If you are simulating DL/I access by using the MERANT products that provide *local* simulation, calls to EZEROLLB cause a loss of database positioning in DL/I, but have no effect on DL/I data.
- If you are accessing DL/I remotely on MVS by using the MERANT products or the VisualAge remote DL/I option, calls to EZEROLLB cause a rollback of changes to DL/I data even when the target environment is VSE Batch, which does not support rollback of DL/I data. The DL/I rollback eliminates the need for manual cleanup of the data.

The DL/I commits and rollbacks are independent of those for SQL databases, and all commits are single-phase commits.

Sharing PSB parts across target environments

A VisualAge Generator PSB part can be shared across multiple target environments, and VisualAge Generator takes the target environment into account when determining the PCB number that a DL/I call references.

VisualAge Generator adjusts for I/O or TP PCBs defined in the VisualAge Generator PSB part if the target environment does not support those PCBs.

If you are going to use the same VisualAge Generator PSB part to test with different target environments, you may need two different IMS PSBs. Define the PSB in the target environment as you normally would. Include no I/O or TP PCB for a PSB in MVS CICS, VSE Batch, or VSE CICS. Include the I/O and TP PCBs in the PSB used for other environments.

Understanding data conversion in the test facility

The test facility assumes that character data is stored in EBCDIC format in DL/I databases. The test facility converts the data to ASCII for use within the test facility, and data defined in the test facility is converted to EBCDIC before being stored.

Whether you access DL/I by way of VisualAge or Micro Focus, and whether you access DL/I remotely or by a local simulation, segments are retrieved from the database in EBCDIC collating sequence. Information retrieved in the EZE word EZEDLPCB is also converted to ASCII.

A data-conversion table guides ASCII/EBCDIC conversion and binary-data conversion. On workstations, the default table name is the setting of environment variable EZERCVT. If EZERCVT is not set, the default conversion-table name is ELACNxxx, where xxx is the value of environment variable EZERNLS and is, by default, ENU. For further details on data conversion, see *VisualAge Generator Client/Server Communications Guide*.

Passing DL/I data in the test facility

If your test involves multiple programs accessing DL/I data, do not run a mixture of generated programs and test programs. Make sure all of the DL/I programs are run from the test facility.

Chapter 4. Developing segmented programs

During program definition, you can specify the runtime mode for a program. The program can run in any of the following modes:

- Segmented
- Single-segment
- Nonsegmented

MVS/TSO, VM CMS, OS/2, Windows NT, AIX, HP-UX, Solaris, or test facility

The logical effects of segmented mode are simulated by committing recoverable resources on a segmented CONVERSE.

CICS Nonsegmented is equivalent to CICS conversational processing, while segmented or single-segment modes are equivalent to CICS pseudoconversational processing.

IMS Programs must always run in segmented or single-segment mode.

Running in segmented mode

When running in segmented mode, a program saves current program status in a work file or database. An example of current program status is the current values for variables. The program releases all storage, file, and database resources whenever it requests input from the terminal using a CONVERSE statement or transfers with a map using an XFER statement. When running in nonsegmented mode, these resources are not released.

Before defining programs that run in segmented mode, you must understand the effect of segmenting in the runtime environment. Because segmenting might alter the results of the program, you must consider whether or not to segment programs during the initial design phase.

Segmented mode enables a larger number of terminals to run VisualAge Generator programs within the same system storage address space for CICS systems at the same time. Although segmenting programs enables concurrent use by a larger number of terminals, the response time for each terminal is increased by the time required for each transfer of data (roll out or roll in), and the time required by the host subsystem to create a new system task.

Segmented programs do not use address space during user think time. This is because the program address space is saved on external storage when the current system task ends, and the program needs input from the user to continue.

When the user presses **Enter**, **Clear**, a PA key, or a function key, a new system task is started. This task's address space is restored by VisualAge Generator with the data retrieved from external storage.

Running in single-segmented mode

Use single-segment mode if you want to reduce the amount of information saved during the program *intermessage delay time*, that is the time between receipt of a system response at a terminal and the time when a new transaction is entered, better known as *user think time*. With single-segment mode, you define a program that represents a single runtime segment from program start to end. The program ends with the first XFER statement, when the EZECLOS special function word is invoked, or at the end of the program logic.

The CONVERSE I/O options are not supported in a single-segment program. To read a map at the start of your program, you select the **First Map** option during program specification. At the end of your program, you can use an XFER statement with a map.

Running in nonsegmented mode

Nonsegmented programs consume address space from start to finish including user think time. User think time starts with each map CONVERSE function and varies by program and map. System resources are used by the program while waiting for the user to enter the next transaction, for example, when the user presses **Enter** or **Clear**.

Comparison of segmented and nonsegmented program designs for CICS

You can specify in CICS environments whether a program runs in segmented (CICS pseudoconversational) or nonsegmented (CICS conversational) mode during program specification. You can also dynamically change the runtime mode for the program by using the EZESEGM special function word. EZESEGM is set to the default value (1 for segmented mode and 0 for nonsegmented mode) after every CONVERSE.

Figure 12 on page 123 illustrates the flow of a program running in nonsegmented mode, and Figure 13 on page 124 illustrates the flow of a program running in segmented mode. The sample update program, CSUP used in both figures, converses a map, displays the customer data that can be updated, accepts data from a user to update the customer record, and replaces the record with the changed data.

When you use nonsegmented mode (as in Figure 12), you should set EZECNVCM special function word to 1. This causes a commit point to occur

at the CONVERSE so changes to files and databases are committed and locks are released. Figure 12 also illustrates saving a copy of the record for comparison purposes after the CONVERSE to ensure that no other changes have been made to the record during user think time.

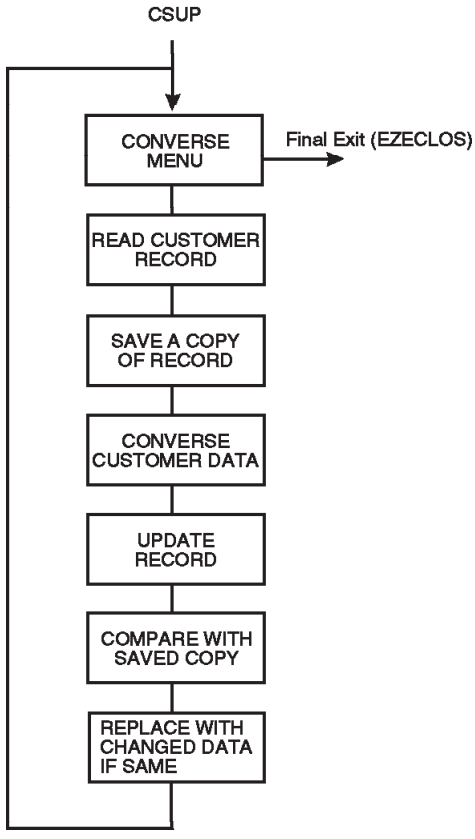

Figure 12. Update File Program Running NonSegmented Mode

Figure 13 on page 124 shows the flow of a program running in segmented mode.

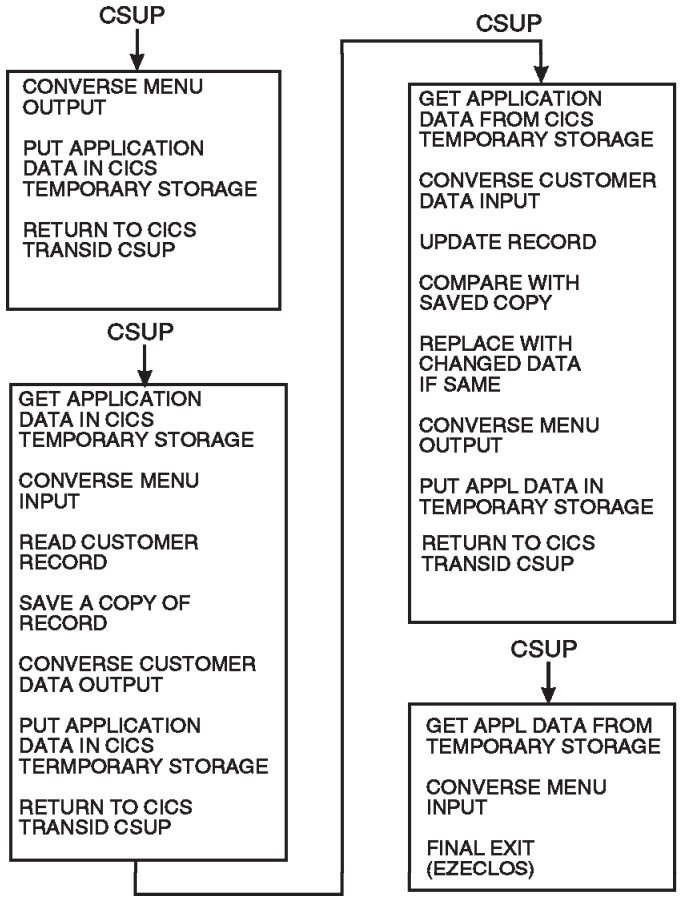

Figure 13. File Program Running in Segmented Mode

When a program runs in segmented mode, temporary storage must be provided to contain the roll out/in data during segmentation. Each program requires approximately 6000 bytes plus the total size of all objects accessed by the program (records, working storage, and maps). For this reason, you might want to make the program, print services program, and map group format module resident for segmented programs.

Choosing between segmented and nonsegmented programs

When you are deciding whether to design your program as segmented or nonsegmented, you should be aware of two issues. The first issue is the effect of the transaction on contention resources, such as storage and processor use. The second issue is the effect on exclusive use resources, such as records and recoverable data sets, recoverable transient data queues, and enqueue items.

Nonsegmented programs have a high impact on storage because they run longer than the sum of the transactions that are in an equivalent segmented program. However, processor overhead is less because only one program is started, instead of one for every transaction.

A nonsegmented program retains exclusive use of resources for a longer period of time, unlike the equivalent segmented program. For this reason, segmented programs are quicker to respond, but for recovery and integrity considerations, you might prefer a nonsegmented program.

If you have maps in called programs or need to lock the database over a CONVERSE, you should design your program to run in nonsegmented mode.

The following list contains considerations for segmented and nonsegmented programs:

- Segmented mode uses more processor time because CICS spends more time initiating and ending transactions.
- Nonsegmented mode uses more virtual storage because transactions are still active during user think time. However, with Dynamic Transaction Routing (CICS/ESA 3.1) CICS can automatically start another region and send transactions to the next region when the first region is constrained.
- Nonsegmented mode can also use other resources such as locks in the database during user think time. (This can be solved by setting the special function word EZECNVCM to 1.)
- CICS accounting and security is less granular with nonsegmented transactions because you have a few large transactions, rather than a lot of small ones.
- CICS shutdown can be more difficult with a lot of nonsegmented transactions. You might have to end transactions before you can shut down because someone is out for a break in the middle of a nonsegmented transaction. (This can be solved by having transactions time out if the user has not pressed **Enter** after a specified time).
- Programming nonsegmented programs can be easier because you can do the following:
  - Use maps in called programs
  - Hold locks and cursor position in the database over a CONVERSE.
- Only segmented transactions can be migrated to the IMS environment.

When running a program on an XA or ESA system where storage contention is not a problem, a good compromise between running in segmented or nonsegmented mode is to run in nonsegmented mode with the EZECNVCM special function word set to 1. This approach forces a commit at every

CONVERSE from the main program, and it has the good performance characteristics of nonsegmented mode, while it does not hold file or database resources during user think time.

Program design Considerations

The following must be considered in designing segmented programs:

- Any called program loses the return point if segmentation occurs; therefore, the following restrictions apply:
  - VisualAge Generator programs defined as called transactions can converse maps but cannot be generated to run in segmented mode. Called transactions are not supported for the IMS environment.
  - If a program is called from a main program that is running in segmented mode, the transaction runs in CICS conversational mode (nonsegmented) until the program returns from the called program.
  - If a VisualAge Generator program calls a non-VisualAge Generator program, the program cannot use segmented mode. Any interaction with the program user must be in CICS conversational (nonsegmented) mode.
- Segmentation ends the current system task. CICS and IMS commit all recoverable resources when the task ends.
- A record cannot be held for update (locked) across a segmented CONVERSE.

  Note: Holding a record for update across a CONVERSE is not a good practice on any system, because it locks resources during user think time, preventing additional users from accessing the system.

  For a better approach to holding a record for update across a CONVERSE, refer to the code in Figure 14 on page 127.

```
PROC1     INQUIRY     RECDA       Get record for compare
   MOVE RECDA TO COMPA;  /*  Save record for compare
                         /*  COMPA is a copy of RECDA
                         /*   in the additional records list
PROC2     CONVERSE    MAPA        Get changes from program user

PROC3     UPDATE      RECDA       Get record for update

   IF RECDA.ALL-DATA NE COMPA.COMPA-DATA;  /* See if changed
                         /*  Build error message and
                         /*   converse MAPA again

PROC4     REPLACE     RECDA    /* Replace record with changes
             .
             .
             .
          RECDA      RECORD
            03 ALL-DATA   CHA n
            05 item-1
            05 item-2
             .
             .
             .
            05 item-n
          COMPA      RECORD
            03 COMPA-DATA CHA n
            05 item-1
            05 item-2
             .
             .
             .
            05 item-n
```

Figure 14. Alternative to Holding a Record for Update across a CONVERSE

- UPDATE locks and SCAN positions in files or databases are lost during a CONVERSE when running in segmented mode.
- The program function and I/O objects determine the amount of response time delay caused by the roll out/roll in process:
  - The longest delay occurs in a segmented program that has a large amount of variable field data on maps or large records and short user think time.
  - The shortest delay occurs in a menu type program that has a small amount of variable field data on maps and long periods of user think time.
- In CICS, if the UCTRAN operand has been specified for the terminal control table (TCT), CICS folds user data from maps when running in segmented mode. The folding of user data prevents users from running with uppercase and lowercase input (Fold=NO).

- On CICS systems, when the user presses **Enter** or a function key, the system returns input data through CICS to the VisualAge Generator program. CICS examines the beginning of the data, searching for Basic Mapping Support (BMS) commands. When designing segmented programs, ensure that the first physical variable field on your VisualAge Generator map does not contain a valid BMS paging command. For more information on design considerations for segmented programs in CICS, refer to the CICS documentation.

Implementing a hierarchical structure for segmenting programs using a DXFR statement

This section describes a set of VisualAge Generator programs that should perform well running in segmented mode in the CICS environment. An additional benefit is that these programs are easier to maintain, test, and enhance than a single VisualAge Generator program that contains all the same functions.

Figure 15 shows the structure of four VisualAge Generator programs that perform three functions. The menu program's only function is to access the three functions. When the user selects the desired function, the menu program transfers control using a DXFR statement to the correct program, passing a small working storage record to further define the request. The transferred-to program prompts the user for required data, performs the function as often as needed and transfers using the DXFR statement back to the menu program.

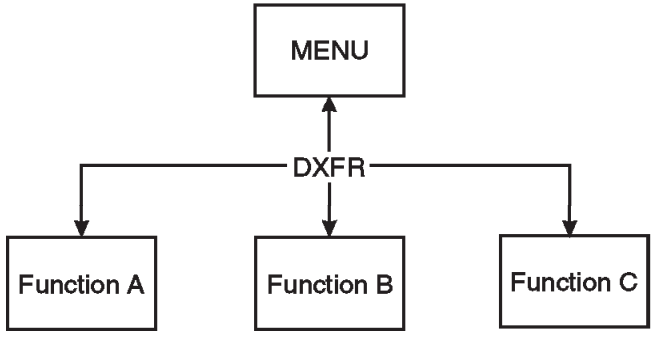

Figure 15. Hierarchical Structure Using a DXFR Statement

The implementation of a DXFR statement solves the following segmented mode restrictions:

- Called programs cannot run in segmented mode.
- Called programs do not release the caller's resources because program control returns to the calling program.
- The amount of data rolled in/out during a segmented converse is much smaller. Only the program currently in control has its data areas saved.

With a DXFR statement in the IMS/VS environment, the storage for the original program (the MENU program in the example), is not released. If you are developing programs that run in both IMS and CICS environments, you can use either a DXFR or an XFER statement. If you are developing programs that run in IMS only, use an XFER statement for the following reasons:

- To free resources for the transferred-from programs
- To cause a commit point and release UPDATE locks
- To permit each program to have its own DB2 plan and a different PSB
- To permit each program to have different performance tuning information in the IMS system definition

Use a DXFR statement in the IMS environment if you do not want a commit point to occur or if you need both programs to use the same DB2 plan and PSB.

Dynamically changing execution mode

The EZESEGM special function word enables the dynamic control of program segmentation at run time. Specifying segmented mode for run time during program specification only sets the default mode for run time. By setting the EZESEGM special function word to either 0 or 1, you can override the default value for any CONVERSE function. Before each CONVERSE, VisualAge Generator Server for MVS, VSE, and VM checks the value of EZESEGM. If EZESEGM equals 1, the CONVERSE is segmented, if EZESEGM equals 0, the CONVERSE is nonsegmented. At the successful completion of a CONVERSE, the system resets EZESEGM to the generated default value. Nonsegmented programs have a default value of 0 for EZESEGM while segmented programs default to 1. This function gives you control to switch in and out of segmented mode for reasons of performance, function, and target system differences. To control segmentation, define the following statements prior to a CONVERSE.

```
MOVE 1 TO EZESEGM ;  /* Force converse to be segmented
MOVE 0 TO EZESEGM ;  /* Force converse to be nonsegmented
```

Remember that EZESEGM is reset to its generated default after every CONVERSE I/O option.

Note: The EZESEGM special function word is ignored for IMS/VS. All CONVERSE functions must be segmented in IMS/VS.

Switching transaction codes for program segments

On CICS and IMS systems, a segmented CONVERSE ends the current transaction. A new transaction is started when the terminal input is received from the user. The new transaction is identified by the EZESEGTR special function word when the CONVERSE I/O option is issued. The default value of EZESEGTR is the current transaction ID associated with the initial program in the current transaction.

If a segmented CONVERSE is used in a program started by a DXFR statement from another program (for example, A transfers using a DXFR statement to B, which issues the CONVERSE), then the default transaction ID starts the original program (A) again on the input from program B's CONVERSE. The generated program reads the transaction status record from the work database, determines that B was the program that issued the CONVERSE, and transfers to B to continue processing. This logic is generated into the program for you.

You can bypass the overhead of restarting the original program by doing the following:

- Defining a unique transaction ID for each segmented program started by a DXFR statement
- Having each program move its transaction ID into EZESEGTR before doing the first CONVERSE.

Each transaction ID you use must be defined to IMS or CICS as being associated with its corresponding program.

The two figures, Figure 16 on page 131 and Figure 17 on page 131, show the differences in program flow for a program that has been transferred-to with a DXFR statement using the default transaction ID and setting the EZESEGTR special function word.

For CICS, if you use the technique in Figure 16 on page 131, which shows a transfer using default transaction IDs, you need one Program Control Table (PCT) entry to associate transaction ABCD with the menu program. If you use the technique in Figure 17 on page 131, which shows a transfer using EZESEGTR, you need two PCT entries, one to associate transaction ABCD with the menu program and one to associate transaction GETD with program GETDATA.

For IMS, if you use the technique in Figure 16 on page 131, which shows a transfer using default transaction IDs, you need one pair of APPLCTN and TRANSACT macros to associate transaction ABCD with the PSB for the menu program. If you use the technique in Figure 17 on page 131, which shows a transfer using EZESEGTR, you need two pairs of APPLCTN and TRANSACT

macros, one pair to associate transaction ABCD with the PSB for the menu program and one pair to associate transaction GETD with the PSB for program GETDATA.

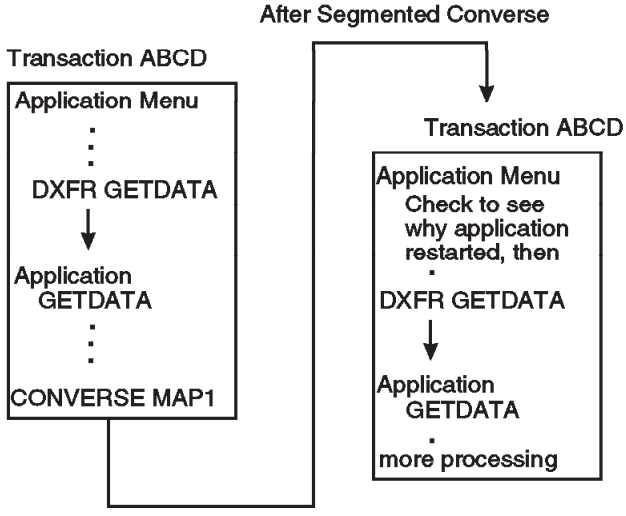

Figure 16. Example Transfer Using Default Transaction IDs

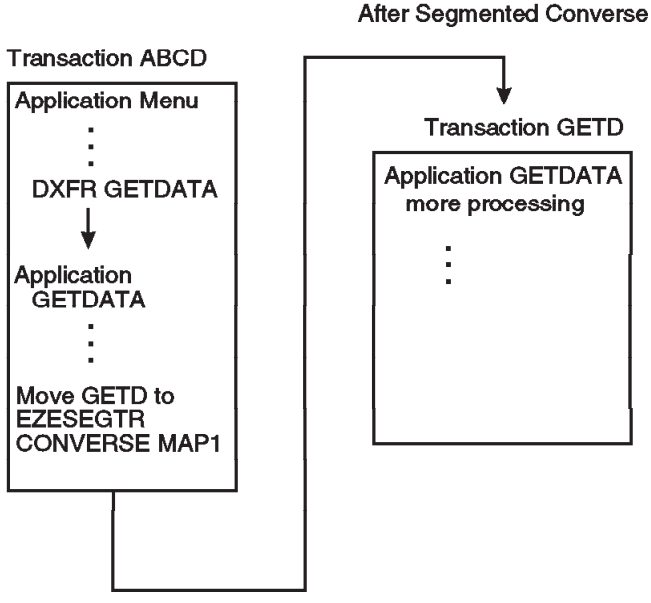

Figure 17. Example Transfer Using the EZESEGTR Special Function Word

Using an XFER statement with map and first map

You can specify a map name on an XFER statement in addition to the record name. When a map name is specified, the generated program displays the map and identifies the next transaction to the CICS or IMS environment. The new transaction is scheduled when input is received from the program user. For the MVS/TSO and VM CMS environments, specifying a map on an XFER statement results in the transferred-from program displaying the map and then transferring to the next program using an OS XCTL macro.

When you use an XFER statement with a map, you must also specify the same map as the First Map for the transferred-to program during program specification. First Map is the name of the map from where the program receives input before the program begins processing. The First Map specification enables the transferred-to program to begin by reading the same map displayed using the XFER statement in the initial program. First Map and XFER with a map enable you to use an IMS deferred program switch or a RETURN TRANSID for CICS.

Note: For IMS/VS, when you use an XFER statement with a map and First Map, the two programs must share the same map group. For other environments, the map can be in different map groups, but it must be the same map.

When you specify First Map, the processing that occurs when a program is started varies:

- If a map is not received, the generated program automatically displays the map that was specified as the First Map.
- If a map is received, the generated program automatically performs any map edits that are required before beginning the normal processing logic.

When you transfer program control using an XFER statement with a map, you control the amount of data saved during user think time and the location where it is saved:

- You can specify a record when you define the XFER statement in addition to the map.

  For the IMS/VS environment, VisualAge Generator Server for MVS, VSE, and VM automatically saves the record in either the Scratchpad Area (SPA), for conversational without the ADF parameter specified on the /SPA generation option, or the work database, for nonconversational or when the ADF parameter is specified on the /SPA generation option.

  For the CICS environment, VisualAge Generator Server for MVS, VSE, and VM automatically saves the record in the COMMAREA.
- You can put additional data on the map by defining map fields having a *dark* attribute so the user cannot see the field. Using a dark attribute allows

the data to be available to the transferred-to program when it reads the First Map, but keeps the user from seeing the data when the map is displayed. A copy of the map is saved in the work database so the data can be displayed again if the user requests a help map.

For the IMS/VS environment, if you want to avoid saving a copy of the map in the work database, you must do the following:

- Set the MODIFY attribute on for all variable fields on the map. You can do this when you define the default attributes for the fields on the map or with the SET statement.
- Ensure that all the other attributes are set to their defined values before the XFER statement.
- You can save data in a database using the VisualAge Generator I/O options and then restore the data in the transferred-to program.
- You can also use an XFER statement to transfer to the same program (Program A can transfer to program A by using an XFER statement).

Accessing multiple DB2 plans in CICS for MVS/ESA

When creating a system of programs that access DB2 tables, you might not want to bind all the database request modules (DBRMs) for each program into one DB2 plan. For security and maintenance reasons, you might want to access several DB2 plans in a system of programs. This section discusses three of the possible methods for accessing multiple DB2 plans in CICS for MVS/ESA. The first two methods describe how to change the transaction ID. The third method uses the DB2 Dynamic Plan Selection function.

If you have associated the DB2 plan name and transaction ID in the CICS for MVS/ESA resource control table (RCT), you can change the DB2 plan name by changing the transaction ID. For more information on the RCT, refer to the appropriate installation or administration manual for your version of DB2.

Accessing DB2 plans using EZESEGTR

The VisualAge Generator special function word EZESEGTR enables you to dynamically change the segmented transaction ID. When running a segmented program, the value in EZESEGTR is used as the transaction ID to start the program again immediately after every CONVERSE. A simple example of using EZESEGTR to dynamically change the transaction ID is described in Figure 18 on page 134.

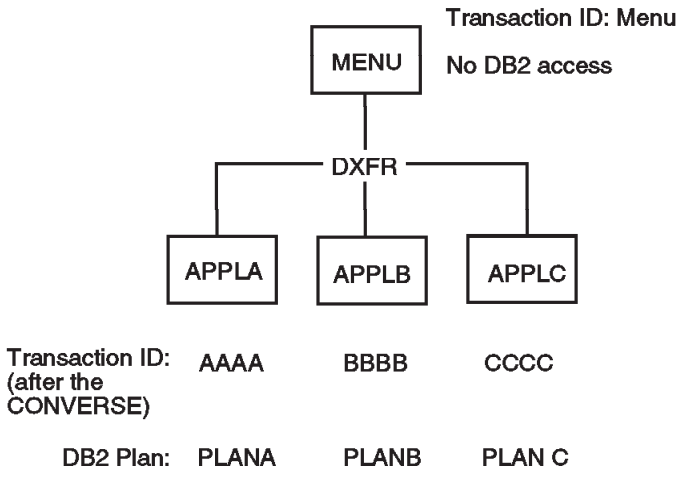

Figure 18. Example of Using the EZESEGTR Special Function Word

In Figure 18, the menu program converses a menu map with three options and issues a DXFR statement to a function program. Each of the function programs moves a transaction ID to EZESEGTR, converses a map, and retrieves a row from a DB2 table. For example, logic similar to the following could be used in each of the function programs:

```
.
.
.
MOVE 'AAAA' TO EZESEGTR ; /*  Set EZESEGTR to a new transaction ID
SHOW_INFOMAP()          ; /*  Show the information map to the user.
READ_DB2_RECORD()       ; /*  Retrieve information from a DB2 table
.
.
.
```

SQL statements are not in the menu program or, prior to the CONVERSE I/O option, in the function programs.

Each function program's DBRM is bound into a unique DB2 plan and associated with a unique transaction ID in the RCT. This is the transaction ID moved to EZESEGTR. After the CONVERSE, a new transaction starts with the plan associated with the new transaction ID.

The transaction IDs AAAA, BBBB, and CCCC are associated in the RCT with DB2 plans PLANA, PLANB, and PLANC, respectively.

This method of association can only be used in segmented programs. However, for CICS programs, you can access DB2 plans with an XFER statement or dynamically select a DB2 plan.

Accessing DB2 plans with an XFER statement

This method is useful if you transfer control between programs using an XFER statement. The transaction ID is changed when you transfer from one program to another using an XFER statement. This gives you access to a new DB2 plan if you associated the new transaction ID with a different DB2 plan in the RCT.

Dynamically selecting a DB2 plan

This method uses DB2 dynamic plan selection, introduced in DB2 Version 2 Release 1, that provides the ability to dynamically select a DB2 plan name for an CICS for MVS/ESA transaction. DB2 dynamic plan selection provides you with the option of defining an exit program in the RCT instead of a DB2 plan name. The exit program selects a DB2 plan for the CICS for MVS/ESA transaction. With this function, you can associate several plans with one transaction ID. For more information about DB2 dynamic plan selection, refer to your DB2 system documentation.

The first SQL statement in a logical unit of work (LUW) starts the exit program.

Accessing multiple DB2 plans in IMS

All the DBRMs that run together using a single IMS PSB must be bound together in a single DB2 plan. The following programs run together using a single IMS PSB:

- A main program and all programs it calls
- A program and all programs that it transfers to using a DXFR statement

You can change DB2 plans with the following techniques:

- Transfer to a new program using a DXFR statement, change the EZESEGTR special function word to a new transaction code, and then do a converse before using any SQL I/O options. For more information on this technique, refer to “Accessing DB2 plans using EZESEGTR” on page 133.
- Transfer control to a new transaction using an XFER statement. Different transactions can use different PSBs and can, therefore, have different DB2 plans.

Error Processing for segmented programs

The test facility issues warning messages if it detects the potential for error. For example, warning messages are issued if a record is currently being updated, a CONVERSE function is encountered, and the EZESEGM special function word equals 1.

The validation and generation step does the following:

- Warns you if the EZESEGM special function word is referenced in a called program. EZESEGM has no effect in a called program.
- Prevents programs being generated for IMS/VS from being nonsegmented.

Chapter 5. Developing IMS programs

This chapter describes design and development considerations for IMS applications. You can perform the following functions using VisualAge Generator to develop IMS programs:

- Define online and batch programs for IMS
- Test IMS programs using the VisualAge Generator Developer test facility
- Generate COBOL programs to run in the IMS/VS or IMS BMP environments

You can convert existing database programs that run in segmented (pseudoconversational) mode in CICS environments or as MVS batch jobs to run in the IMS environment. See "Chapter 4. Developing segmented programs" on page 121 of this document.

See the following documents or sections for additional information on developing programs for the IMS environment:

- "DL/I considerations for non-CICS environments" on page 113
- The following sections in *VisualAge Generator Client/Server Communications Guide*:
  - Implementing client/server processing using the CALL statement
  - Calling IMS servers via APPC
  - Implementing client/server processing using the message queue interface
  - Calls in the MVS/TSO, MVS batch, IMS BMP, IMS, and VSE batch MPP environments
  - Transfers in the MVS/TSO, MVS batch, IMS BMP, and VSE environments
  - Transfers in the IMS MPP environment
  - Interfacing with IMSADF II programs
- "Choosing between CALL and DXFR statements" on page 247
- "Choosing between XFER and DXFR statements" on page 248

Refer to the following VisualAge Generator documents for the following information about programs targeted for IMS/VS or IMS BMP:

- Preparing a program for generation in the *Generation Guide* document.
- Preparing a generated program to run in IMS in the *VisualAge Generator Server Guide for MVS, VSE, and VM* document.
- IMS compatibility considerations by language element in the *Programmer's Reference* document.

Introduction to IMS

IMS is a database/data communication system that can manage a large, complex network of databases and terminals. IMS/VS consists of a database feature (IMS/DB) and a data communications feature (IMS/DC). IMS/ESA consists of a Database Manager (IMS/ESA DM) and a Transaction Manager (IMS/ESA TM). The VisualAge Generator target environments of IMS/VS and IMS BMP support execution on either IMS/VS or IMS/ESA systems. In the discussion that follows IMS is used for both IMS/VS and IMS/ESA, unless explicitly stated otherwise. The terms IMS/VS and IMS BMP are used when a distinction is needed between the target runtime environments.

This section describes program execution, terminal access, and serial file access in the IMS environments. For information about database access, see the "Chapter 2. Developing SQL programs" on page 19 and "Chapter 3. Developing DL/I programs" on page 85 of this document.

IMS/DC and IMS/ESA do the following:

- Bring databases into an online environment so you can have interactive access to them
- Manage transactions and terminals
- Manage the transfer of data between transactions and terminals
- Provide services for displaying information at terminals and printers
- Provide extensive data recovery facilities
- Support programs that require a high volume of transactions and a high rate of data availability
- Provide a link between multiple IMS systems and between IMS and CICS systems through Intersystem Communication (ICS)

Understanding IMS terminology

Before you can develop online and batch programs for IMS/VS and IMS BMP, you need to understand the following terms that have a unique meaning in the IMS environment:

Message processing programs (MPPs)
IMS programs that process requests from terminals and from other programs. The requests or messages are stored in message queues accessible to the MPP. The MPP can perform required database access and write new messages to output queues for further terminal and program processing. MPPs cannot access OS/VS or GSAM files. There are two types of MPPs:

Conversational MPP
A type of MPP that saves data in a scratchpad area (SPA) during user think time, even though locks on the database are lost. IMS conversational mode is similar to CICS

pseudoconversational mode. IMS has no capability that is similar to CICS conversational mode.

Nonconversational MPP
A type of MPP that can process a single input message with, at most, a single response. No data can be saved during user think time (except for data saved on the screen or in a database).

Batch message processing programs (BMPs)
Programs that run as batch jobs but access databases that they share with online transactions. BMPs can access message queues like MPPs for batch processing and can also access operating system files. You start BMPs using JCL. There are 2 types of BMPs:

Transaction-oriented BMP
A BMP that accesses the message queue for input. It can also access databases and operating system files. Output can be sent to databases, operating system files, or a message queue. Only one input file can be associated with the message queue.

Batch-oriented BMP
A BMP that does not access a message queue for input. Databases and operating system files are available for input and output processing, and message queues are available for output.

Batch programs
Programs that can access private databases and operating system files directly. Batch jobs do not access message queues or databases shared with an online system. In IMS documentation, these are referred to as DL/I batch jobs.

BTS Batch Terminal Simulator- An IBM product that enables you to run IMS database and data communication programs in an MVS/TSO or batch environment. BTS provides a comprehensive way to check program logic, IMS program interfaces, teleprocessing activity, 3270 format control blocks, and database activity.

DBD The database definition of DL/I, fast path, or GSAM databases. You specify segment content, hierarchy, and physical characteristics such as organization and access method, in the DBD.

Fast path
A type of IMS processing that involves expedited handling for certain transactions and supports special databases designed for large volumes of data with a high availability rate. There are two types of fast path databases:

DEDB Data Entry Database- A type of IMS fast path database that

contains large volumes of data with a high rate of availability. Subset pointers help manage long chains of segment occurrences. One segment type is stored near the root segment and the occurrences are in chronological order.

MSDB

Main Storage Database- A type of IMS fast path database that uses fixed-length root segments that reside in virtual storage for quick access. The segments can be related to a specific terminal or be defined so all terminals can access the data.

GSAM

Generalized sequential access method- GSAM enables MVS batch programs and BMPs to access a sequential OS/VS data set as a database. The database is a root-only database, and the entire root segment represents a record. Unlike sequential OS/VS data sets, you can checkpoint and restart GSAM files just like DL/I databases.

Message Queue

A place in IMS where information being sent to an alternate terminal or to another transaction can be stored so IMS can handle the I/O to the terminal or schedule the other transaction to start processing. It is accessed through DL/I calls.

MFS Message Format Services- An editing facility that you use with IMS that permits programs to access message data from a terminal. MFS enables you to customize the presentation of the data, but shields the program from panel formats and device dependencies by providing access to only the data required from the terminal.

PCB Program Communication Block- In IMS, a collection of information related to an IMS resource that a program can use. IMS uses this control block to determine the resource being used and to return the results of an I/O operation against the resource to the program.

IMS uses the following types of PCBs:

I/O Represents an IMS logical terminal or a message from another program. A program uses DL/I calls for this PCB to read input from a terminal or program and write output messages back to that same terminal.

Alternate

Represents the message queue for an IMS logical terminal or an alternate transaction. It differs from an I/O PCB because the alternate PCB can represent logical terminals other than the terminal from where the input message came. An alternate PCB is also known as a teleprocessing (TP) PCB.

Express
An alternate PCB that sends a completed message immediately to its destination. A non-express alternate PCB does not send a completed message to its destination until a commit point. When this process is not successful, complete messages sent to an express PCB cannot be backed out. A message is complete when a PURG call is issued. You can force a PURG call by using the VisualAge Generator CLOSE I/O option. An express PCB is also known as a teleprocessing (TP) PCB.

DB Represents a DL/I database that a program can access. In addition, the DB PCB specifies the data that the program can access, the segment or field level, and the type of processing valid with that database.

GSAM
Represents a GSAM file that a program can access and contains the processing option available for the program.

PSB A Program Specification Block- In IMS, a PSB is a set of statements that define the PCBs a program can use. The database PCBs identify the required databases, segments to be accessed, and database options for a given program. Alternate PCBs define message queues for terminals or other programs.

Program switch
A way of transferring control from one program to another. There are two types of program switches:

Deferred program switch
Occurs when Program A responds to the terminal and informs IMS to start another transaction that is associated with Program B on the next input from the terminal.

For a conversational MPP, the program switch is done by modifying the SPA to specify the new transaction name before sending it back to IMS through the I/O PCB.

For a nonconversational MPP, the program switch is done by including the next transaction name on the map so it is the first 8 bytes of the input message.

Immediate program switch
Occurs when Program A passes control directly to another transaction that is associated with Program B without responding to the originating terminal first.

For a conversational MPP, the program does this by inserting the SPA to an alternate PCB that has its destination set to the new transaction name.

For a nonconversational MPP, the program inserts a message to an alternate PCB that has its destination set to the new transaction name.

SPA Is the scratchpad area (SPA) for IMS. The SPA is used for conversational processing to save data while the map is displayed during user think time.

Work database
Is a database that is used to save information about a running program during user think time. Except for defining a PCB for the work database or including it in your DB2 plan, you do not need to do any special processing for a work database. VisualAge Generator Server for MVS, VSE, and VM manages and uses the work database for you. The work database can be a DL/I database or a DB2 database.

Interacting with terminals in IMS

Typical IMS programs use a message-driven structure like the example shown in Figure 19.

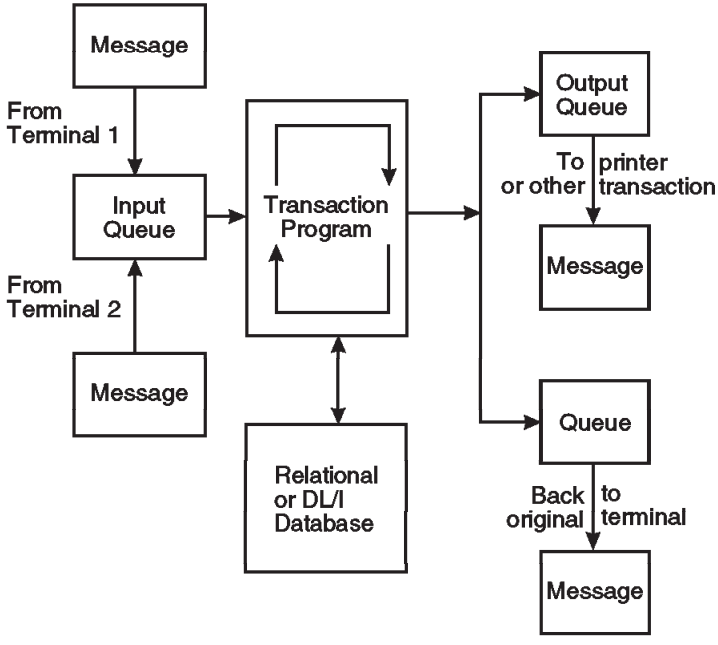

Figure 19. IMS Program Development Considerations

In this example, the IMS controller starts the transaction program when the message queue associated with the program contains a message. The message might have been put on the queue by another program or as a result of the controller reading input from the terminal. The program takes the message off the queue, does any required database I/O, and adds messages to output queues to continue further processing. The output queue can represent the input terminal, another terminal or printer, or a queue associated with another transaction. The program then loops back to the beginning and processes the next message on its input queue.

Typical PL/I or COBOL programs must continue the cycle until the message queue is empty because multiple terminals can be running the same transaction concurrently. However, with VisualAge Generator main transaction programs, the loop to read the next message in the queue is automatically handled. You do not need to define message queue control functions directly. You can define programs for IMS just as you define programs for CICS, that use a synchronous logic structure instead of a message-driven structure. The Figure 20 shows an example of a synchronous program.

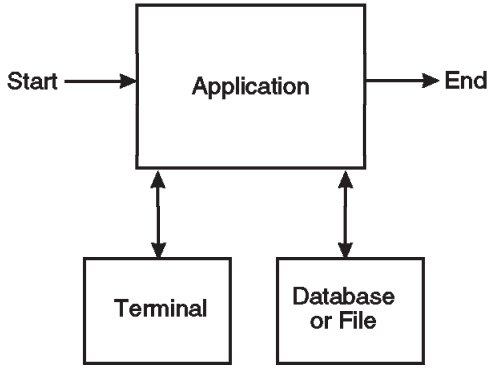

Figure 20. Synchronous Program

With the synchronous model, you only need to consider the processing that must occur for a single user at a single terminal. This simplifies both the design and the definition of the program.

IMS requires that database changes are committed and that database locks and positions are released when waiting for user input. In VisualAge Generator, *segmented* is the term for this mode of operation. When you are defining the program, remember that a commit is performed each CONVERSE I/O option. You must understand how segmentation works to develop programs for IMS. For design and development considerations, see "Chapter 4. Developing segmented programs" on page 121 of this document.

IMS program development methods

VisualAge Generator programs can interact with users at terminals in IMS in either of the following ways:

- An XFER statement with a map and First Map that can be used in either segmented or single-segment mode
- CONVERSE I/O option

This section describes the use of a CONVERSE I/O option or an XFER statement with a map in different types of IMS programs. For more information on using an XFER statement with a map and First Map, see "Using an XFER statement with map and first map" on page 132.

An IMS conversational program

If you want to define an IMS conversational program, you can define a VisualAge Generator main transaction and use the CONVERSE I/O option for terminal I/O. During program definition, you specify segmented execution mode. You must also specify the /SPA generation option. The generated program runs as an IMS conversational message processing program.

For conversational programs, the following functions are automatically handled by the generated code:

- Reading the message queue to obtain the SPA and the map input.
- Saving program information that is needed to resume processing after a CONVERSE function. This information is saved in the work database.
- Writing the SPA and map output to the message queue to continue the conversation.

An IMS nonconversational program in segmented mode

Developing IMS nonconversational programs in this mode is very similar to developing IMS conversational programs. If you want to define an IMS nonconversational program, define a VisualAge Generator main transaction. You can use the CONVERSE I/O option for terminal I/O. During program definition, you specify segmented execution mode. The only difference from developing a conversational program is that at generation time, you must specify /SPA=0 as a generation option. The generated program runs as an IMS nonconversational message processing program.

For nonconversational programs, the following functions are automatically performed by the generated code:

- Reading the message queue to obtain the message or the map input.
- Saving program information that is needed to resume processing after a CONVERSE function. This information is saved in the work database.
- Writing a map to the message queue to continue processing.

An IMS nonconversational program in single-segment mode

The amount of information saved at a CONVERSE function can be quite large. It includes all the records and maps used by the program, as well as the EZE special function words and information about where processing of the program must resume. If you want to reduce the amount of information saved during user think time, you can develop programs using single-segment mode.

With single-segment mode, you define a program that represents a single execution segment from program start-up to program end. The program ends with the first XFER statement, when the EZECLOS special function word is encountered, or at the end of the program logic. CONVERSE I/O options are not supported in a single-segment program.

With single-segment mode, you must use the First Map option to read a map at the beginning of your program. You use an XFER statement with a map to write a map at the end of your program.

Because you control the data that is saved and where it is saved, you can minimize the amount of overhead involved. Therefore, single-segment programs provide the highest level of performance for IMS programs.

Single-segment main transactions without a First Map are supported only if they are the object of a transfer by an XFER statement without a map.

An IMS nonconversational program using batch programs

If you have a nonconversational program that does not use maps to communicate with a terminal, you can define a VisualAge Generator main batch program that processes the message queue. In this situation, you use the SCAN I/O option to read records in a serial file. At generation, you associate the serial file with the IMS message queue and the I/O PCB. As in a batch program for other environments, you must continue processing the input file until it is empty.

If a program transfers to a batch program using an XFER statement with a record, the transferred record is not used to initialize the transferred-to program's working storage record. Instead, the batch program must read the transferred record by using a SCAN I/O option.

Developing IMS fast path programs

You can develop IMS fast path programs using any one of the nonconversational development examples previously described. However, segmented mode (including CONVERSE functions) is not recommended for IMS fast path programs for performance reasons.

IMS imposes restrictions on fast path programs. These restrictions result in the following limitations on the use of VisualAge Generator functions with fast path programs:

- An XFER statement without a map is supported only to a non-fast path program. In this case, the transferred-to program is responsible for responding to the terminal. An XFER statement with a map or both a map and working storage is permitted.
- CSPTDLI is limited to using the call types supported for fast path transactions.
- Multiple-segment input message queues are not supported.
- Only one of the following actions can be done for each get unique to the I/O PCB:
  - XFER statement
  - ADD function for serial file associated with an alternate response PCB
  - CSPTDLI service routine call using the I/O PCB or an alternate response PCB

To indicate that you want a nonconversational program to run as an IMS fast path program, use the /FASTPATH generation option. This option causes the generated program to limit its use of IMS functions to that permitted by IMS fast path support. Use of fast path restricts the amount and type of diagnostic data that is provided when an error occurs in the generated program.

When a batch program runs as IMS BMP and updates IMS fast path databases, the program must explicitly issue either a SYNC or CHKP call to commit the updates. You can force a CHKP call by:

- Using the EZECOMIT service routine before the end of a batch-oriented BMP
- Making sure that the SCAN function for a serial file associated with an IMS message queue receives an EOF (QC status code) before the end of a transaction-oriented BMP.

Sample program flow

You can develop programs for IMS using one of the following techniques or a combination of them:

- Segmented processing using a CONVERSE I/O option
- Segmented processing using a CONVERSE I/O option with the EZESEGTR special function word
- An XFER statement with a map and First Map.

Figure 21 on page 148, Figure 22 on page 149, and Figure 23 on page 150 contain examples of a program's runtime flow using these techniques. The sample program or series of programs that are used in the following figures

converse a map, display the customer data that can be updated, accept data from a user to update the customer record, and replace the record with the changed data.

The first two figures, Figure 21 on page 148, Figure 22 on page 149, use one program running in segmented mode. The only difference in program design is that Figure 22 on page 149 uses the EZESEGTR special function word to have multiple transactions associated with a single VisualAge Generator program. When the converse is done, the data on both the MENU and CUSTOMER maps, as well as the CUSTOMER record, must be saved.

Figure 23 on page 150 shows the use of two single-segment programs to accomplish the same function. The same map must be specified on the XFER statement and as the First Map of the transferred-to program. The same map group must be specified for both the transferred-from and transferred-to programs. With single-segment mode, you control the data to be saved during user think time. In this case, only a copy of the customer record needs to be saved. Single-segment mode enables you to minimize the amount of data that must be saved; therefore, it can improve performance.

Note: DXFR can be used to transfer between programs without changing the IMS transaction. See "Implementing a hierarchical structure for segmenting programs using a DXFR statement" on page 128 and "Choosing between XFER and DXFR statements" on page 248 for information about situations that you might want to use a DXFR statement instead of an XFER statement.

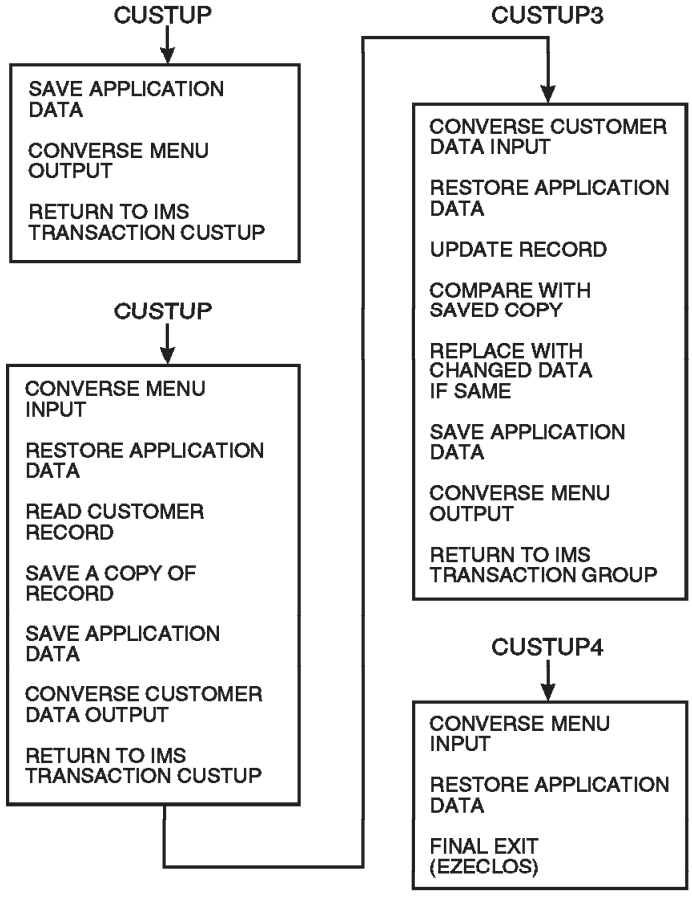

Figure 21. Segmented Mode Technique

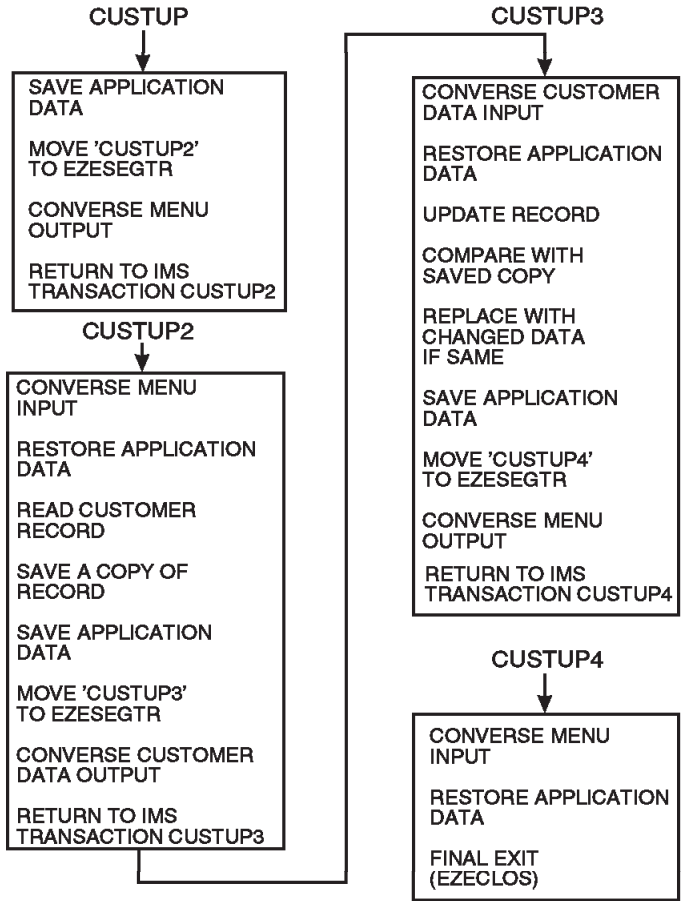

Figure 22. Segmented Mode Using EZESEGTR Special Function Word

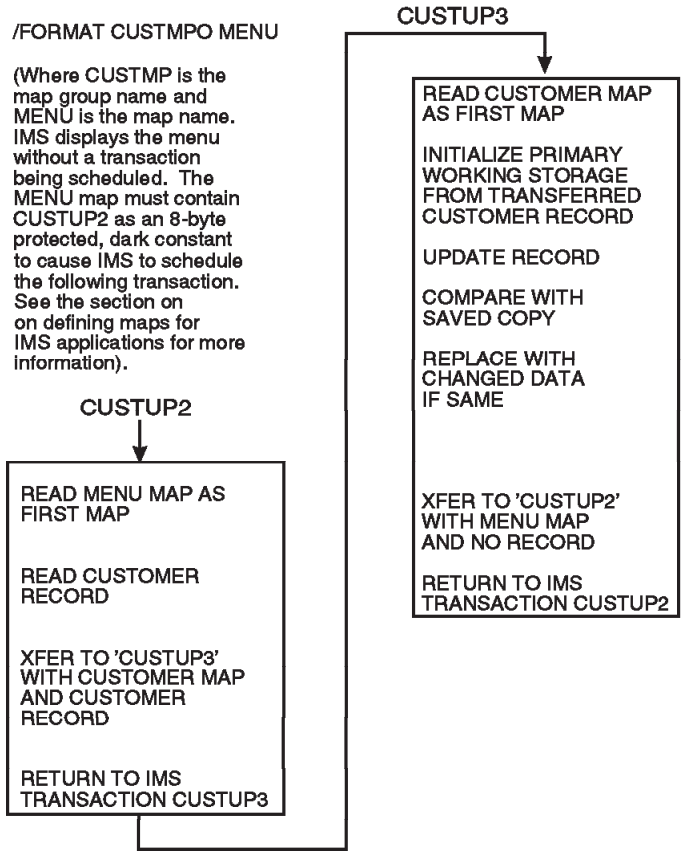

Figure 23. Two Single Segment Programs

Defining data in IMS programs

For IMS programs, you define data the same as you define data for any other type of program. Only PSBs are defined differently for the IMS environment.

Defining PSBs

IMS uses PSBs and PCBs in a different way than other systems. An IMS PSB is a set of statements that define the PCBs a program can use.

An IMS PCB is a collection of information related to an IMS resource that a program can use. IMS programs use PCBs to do the following:

- Receive message and device input (the I/O PCB)
- Deliver message and device output (the I/O and alternate PCBs)
- Define the structure of DL/I databases (DB PCBs)
- Provide serial file support (GSAM PCBs for MVS batch and IMS BMP only)

VisualAge Generator uses the database PCBs in all environments to create default segment search arguments (SSAs). VisualAge Generator supports the presence of all types of PCBs for IMS programs. The VisualAge Generator PSB definition must match the IMS PSB definition exactly.

You need to generate an IMS PSB to correspond to the VisualAge Generator PSB. For IMS/VS, the IMS PSB must have the same name as the load module for the associated COBOL program. An application control block (ACB) generation is also required for the IMS/VS environment. For IMS BMP and DL/I batch, the IMS PSB name does not have to match the program load module name.

You can define VisualAge Generator PSBs using two methods in VisualAge Generator Developer. How the VisualAge Generator PSB is stored and how you can access it depend upon the method you use to create the PSB.

You can define a VisualAge Generator PSB using **PSB Definition**. Because the definition is local to the VisualAge Generator PSB, you can change it using VisualAge Generator Developer. VisualAge Generator PSBs defined using **PSB Definition** cannot be shared with other tools enabled to TeamConnection.

You can also create a new VisualAge Generator PSB by associating it to an existing IMS PSB object. See "Sharing IMS PSBs in TeamConnection" on page 154 for more information.

When you define the PSBs for IMS programs, consider the following criteria:

- The I/O PCB is automatically supplied and does not appear in the IMS or VisualAge Generator PSB source. For IMS/VS or IMS BMP execution, the IMS PSB source must indicate CMPAT=YES.
- Alternate PCBs are used to route output to terminals other than the originating terminal, or to other transactions. Alternate PCBs must appear before the database PCBs both in the IMS and the VisualAge Generator PSB source.
- Batch programs that support DL/I, both in IMS BMP and MVS batch target environments, can implement serial files as GSAM databases. These GSAM files are treated as a special type of database and require a PCB in the PSB. The GSAM PCBs must follow all database PCBs.
- When a VisualAge Generator program is generated for the IMS/VS or IMS BMP environment, a modifiable alternate PCB and a modifiable express alternate PCB are required, in that order, as the first two PCBs following the I/O PCB. Both of these PCBs must have the parameters ALTRESP=NO and SAMETRM=NO. The PSBGEN statement must include the parameters CMPAT=YES and LANG=COBOL or LANG=ASSEM. To avoid having to edit your DL/I call modifications to adjust for the two

required PCBs, include these PCBs whenever you plan to generate a program for the IMS/VS or IMS BMP target environments.

- If you are creating PSBs to use with programs that you plan to generate for the MVS/TSO or MVS batch environments, you must include at least two PCBs of any type in the PSB. The PSBGEN statement must include the parameters CMPAT=YES and LANG=COBOL or LANG=ASSEM.

If a DL/I work database is used, the PCB for this database must be included in the IMS PSB. This PCB can be created using the macro ELAPCB and concatenating ELA110.ELASAMP as part of the SYSLIB in the PSBGEN procedure. Figure 24 shows an example of the PCB expansion that occurs when ELAPCB is used. WORKDBD defaults to ELAWORK. The WORKDBD parameter must be used if the DBD name is changed.

```
ELAPCB   [WORKDBD=customer-dbd-name]

 --- expands into ---

PCB    TYPE=DB,DBDNAME=customer-dbd-name,PROCOPT=AP,KEYLEN=19
SENSEG NAME=ELAWCNTL,PARENT=0
SENSEG NAME=WORKLV01,PARENT=ELAWCNTL
SENSEG NAME=WORKLV02,PARENT=WORKLV01
.
.
.
SENSEG NAME=WORKLV14,PARENT=WORKLV13
SENSEG NAME=MSGLV01,PARENT=ELAWCNTL
SENSEG NAME=MSGLV02,PARENT=MSGLV01
.
.
.
SENSEG NAME=MSGLV14,PARENT=MSGLV13
```

Figure 24. Generating the DL/I Work Database PCB

Defining PCBs

VisualAge Generator supports the following types of PCBs for use with IMS programs that access DL/I databases:

TP Is used to represent an alternate PCB. You must define one TP PCB for each alternate PCB that exists in the IMS PSB. For TP PCBs, the **Type** field is required. All other fields must be blank.

You must define at least two TP PCBs in PSBs for use in the IMS/VS and IMS BMP environments. The first TP PCB must be a modifiable alternate PCB that is used for transaction switching. The second TP PCB must be a modifiable express alternate PCB that is used for

diagnostic information. These two TP PCBs are required. You can define additional modifiable or non-modifiable alternate or express alternate PCBs.

You do not define the I/O PCB in either the IMS PSB or the VisualAge Generator PSB. To include the I/O PCB in the PSB, you specify CMPAT=YES on the IMS PSB definition.

TP PCBs are not used in other environments, but they can be included for compatibility with IMS. In the other environments, the TP PCBs are place holders when determining the offset of database PCBs in the PSB.

DB Is used to define one database PCB for each database that is in the PSB. If you are using DL/I to implement the work database used by VisualAge Generator Server for MVS, VSE, and VM, the PCB for this database should follow the database PCBs for the program. See "Defining a PCB for the work database" for more information.

GSAM
Is used to define a GSAM PCB. When you use a GSAM PCB, **Type** and **Database** are required fields in the VisualAge Generator PSB definition. All the other fields must be blank.

For programs generated for MVS/TSO or MVS batch, at least 2 PCBs of any type are required in the PSB.

You can define PSBs so they can be used in other environments with some restrictions. If the same program and PSB are used in other environments as well as in IMS/VS and IMS BMP, you should include any TP PCBs that are used by the program in the IMS/VS or IMS BMP environments.

Numbering PCBs

Some VisualAge Generator functions require you to specify a PCB number. Some of the functions that require you to specify a PCB number are DL/I call definition, the EZEDLPCB special function word, and the resource association file. The PCB number is the relative number of the PCB as shown in the PSB Definition window. The I/O PCB, that does not appear in the definition, is always number zero.

Defining a PCB for the work database

VisualAge Generator programs running in IMS use a work database to save information that is required to resume processing when running in segmented mode. Your system administrator specifies whether this database is implemented as DL/I or DB2 in the generation options file for VisualAge Generator Developer. If DL/I implementation is specified, you must include a PCB for the database in both the IMS PSB and the VisualAge Generator PSB definition.

You can include the work database in VisualAge Generator PSB definition by setting the **Type** field to **DB** and the database name to ELAWORK. The **Segment**, **Parent**, and **Index key** fields must be blank.

If you do not include the work database in the original PSB definition but add it later, you must add it after the last database PCB but before any GSAM PCBs. This prevents you from having to modify the PCB number specified in any DL/I call modifications or the subscript used with the EZEDLPCB special function word for DL/I databases. However, you must modify the PCB number specified for any serial file implemented as a GSAM file during generation when resource association information is specified, as well as the subscript used for the EZEDLPCB special function word when it references any GSAM file.

Note: If you plan to use a DL/I work database with VisualAge Generator Server for MVS, VSE, and VM in the future, you can avoid making DL/I call modifications or other changes to the PCB number by including place-holder PCBs in both the IMS and the VisualAge Generator PSB. These place-holder PCBs can reference a DBD for a root-only database that is never used by the program.

Sharing IMS PSBs in TeamConnection

You can create a new VisualAge Generator PSB by associating it to an existing IMS PSB object. In this case, the VisualAge Generator PSB shares the existing IMS PSB definition that was created by using DataAtlas. If the IMS PSB definition is changed in DataAtlas, the change is reflected in VisualAge Generator Developer. Because VisualAge Generator Developer does not support the creation or maintenance of IMS PSBs, you can only view VisualAge Generator PSBs that are associated to IMS PSBs.

You can create a new VisualAge Generator PSB that is associated to an existing IMS PSB by using **PSB with Association**. On the PSB with Association window, you can use **Display** to list the IMS PSBs in the library and display the definition stored in an IMS PSB. You can then create a new VisualAge Generator PSB that shares this definition.

You can change the association of a VisualAge Generator PSB to a different IMS PSB by using Change Member Association. On the Change Member Association window, you can use **Display** to list the IMS PSBs in the library and display the definition of an IMS PSB. You can then change the association of a VisualAge Generator PSB to another IMS PSB.

Defining maps for IMS programs

Each terminal map in the IMS environment must contain an 8-byte constant field. This field is used to store the IMS transaction name in the MFS definition for the map. You can define this field as an 8-byte constant field with the protect and dark attributes. The attribute byte on the map becomes the attribute byte in the generated MFS. The 8-byte constant contains the name of the IMS transaction that is started when the map is processed. Specifying the constant on the map enables the user to specify the IMS /FORMAT command to display a formatted screen to start a transaction. The /FORMAT command should not be used if variable fields on the map contain default data. If the /FORMAT command is used, the default values do not appear.

If you do not define an 8-byte, protected, dark constant on the map, VisualAge Generator Developer searches for any string of 9 blanks on the map and sets this area aside as a protected, dark variable field (1 byte attribute, 8 bytes of data) in the generated MFS map. The generated program uses this field to store the name for the next IMS transaction to be run after a CONVERSE function or an XFER statement with a map. The user cannot use the /FORMAT command to start a transaction for these maps because IMS does not have a default transaction name.

An additional unused 2-byte field is also required on each terminal map. VisualAge Generator Developer selects a 2-byte blank field on the map and treats it as a protected, dark variable field (1 byte attribute, 1 byte of data). The field is used to indicate the type of information stored in the work database.

Estimating the size of MFS blocks for a map group

When a VisualAge Generator map group is generated, MFS control blocks are generated for the map group. There are three types of control blocks:

- Device input format (DIF) and device output format (DOF). These control blocks describe the arrangement of data fields and literals on the device presentation space (for example, the screen for 3270 devices).

  For 3270-type devices a single set of statements describe both the DIF and the DOF. For printers, only a DOF is needed. Each device field is given a name that can be referred to by statements in the message input and output descriptors.

  For VisualAge Generator map groups, the DOF is always larger than the DIF because the DOF includes map constants.
- Message output descriptor (MOD). This control block describes the various fields of information in the output message inserted by the program. It also identifies corresponding device fields where the data for each message field is moved.

- Message input descriptor (MID). This control block describes the various fields of information in the input message retrieved by the program. The MID identifies the corresponding device field from where the data for each message field came from.

MFS control blocks cannot exceed 32748 bytes. If you are using a large map group, use the following formulas as a guideline for estimating an upper limit for the size of the control blocks that will be generated. Using these formulas during your design helps you determine whether map groups should be split into smaller ones. If a generated control block is too large, MFS generation issues a 3022 abnormal termination.

Calculating the DOF size for terminal devices

The following formula helps you estimate the size of the DOF.

```
DOF Size =

   150
 + 388 *  Number of printer maps in the map group
 + 208 *  Number of terminal maps in the map group
 +  63 *  Number of map variable occurrences on terminal
          maps in the map group
 +  62 *  Number of map constants on terminal maps in the
          map group
 +1.12 *  Total length of all map constants on terminal maps
          in the map group
```

Calculating the DOF size for printer devices

The following formula helps you estimate the size of the DOF.

```
DOF Size =

   206
 +  68 *  Number of printer maps in the map group
 + 374 *  Number of terminal maps in the map group
 +  63 *  Number of map variable occurrences on printer
          maps in the map group
 +  62 *  Number of map constants on printer maps in the
          map group
 +1.12 *  Total length of all map constants on printer maps
          in the map group
```

Calculating the MOD size for terminal maps

The following formula helps you estimate the size of the MOD.

```
MOD Size =

    36
 + 724 *  Number of terminal maps in the map group
 + 202 *  Number of printer maps in the map group
 +  52 *  Number of map variable occurrences in the map group
```

Calculating the MID size for terminal maps

The following formula helps you estimate the size of the MID for terminal maps.

```
MID Size =

     36
  + 858 *  Number of terminal maps in the map group
  +  52 *  Number of map variable occurrences for terminal maps
           in the map group
```

Defining IMS programs

You can define programs for IMS the same as you define other non-IMS VisualAge Generator programs. There are some exceptions if you use serial and printer files in IMS programs.

Using service routines

You can use the CREATX service routine and the AUDIT service routine in IMS/VS and IMS BMP environments.

You can also use the CSPTDLI service routine that enables you to issue any DL/I call that is supported by the run-time environment. You can use CSPTDLI for DL/I database calls that are not supported by VisualAge Generator I/O options (for example, the FLD and POS call) or to perform functions that are not directly supported by VisualAge Generator (for example, symbolic checkpoint and restart for an IMS BMP). For more information about the CREATX, AUDIT, and CSPTDLI service routines, refer to the *Programmer's Reference*

Using serial and printer files in IMS programs

Serial files must be implemented as IMS message queues in IMS/VS. They can be implemented as message queues, OS/VS files, VSAM files, or GSAM files for IMS BMP. Serial files can be implemented as OS/VS files, VSAM files, or GSAM files for MVS batch. The following sections describe how to use GSAM files or message queues for serial files.

Using serial files as GSAM files

VisualAge Generator programs that run in the IMS BMP or MVS batch environments can implement serial files as GSAM files. You can use the ADD, SCAN, and CLOSE I/O options for serial files that you implement as GSAM files. The following list describes the differences between GSAM and normal serial file processing:

- A GSAM file requires a DBD.
- A GSAM file requires a PCB in the IMS PSB. You must define this PCB in the IMS PSB and in the VisualAge Generator PSB definition. You must include the VisualAge Generator PSB name during program specification.

- A GSAM file is read or written through DL/I calls. The generated COBOL program handles this automatically, based on the I/O options that you request.
- A GSAM file is checkpointed and restarted in the same way as a DL/I database. However, to recover the GSAM file requires the use of symbolic checkpoint and restart instead of basic checkpoint.

VisualAge Generator does not support the record search argument for GSAM or undefined length records.

You identify a serial file or printer file as a GSAM file by using the resource association file during generation to specify a file type of GSAM and a PCB number.

When you associate a serial file with a GSAM file, you must include the following information:

Resource name
Indicates the 1- to 44-character data set name that is used in the sample runtime JCL. The file name is used as the DD name in the sample runtime JCL.

File type
Specifies GSAM as the file type to associate the serial file or printer output with a GSAM file.

PCB number
Specifies a PCB number for the serial file that is associated with the GSAM file. If you do not specify one, the default is the first GSAM PCB in the VisualAge Generator PSB.

Using serial files as message queues

Online programs that run in IMS/VS implement serial files as IMS message queues. Programs that run as IMS BMP programs can also implement serial files as message queues. You can use the ADD and SCAN I/O options as well as CLOSE for output files. If you select IMS/VS or IMS BMP as the target runtime environment, you can define serial or print files as being associated with a message queue. You must associate all serial files and print files with message queues for IMS/VS. Only a single input file can be associated with the message queue.

You can associate a serial file or printer file with a message queue by using a resource association file during generation and specifying the file type and a PCB number. When you associate a serial file with a message queue, you must define the following resource information:

Resource name
You must indicate the 1- to 8-character destination ID for printer or

serial file data. The name must match the ID of an IMS logical terminal or a transaction code that is defined in the IMS system definition.

The file name is the default resource name for the message queue. You can override this default in the resource association file.

You can also override the default message queue name at run time. If the PCB that you select is a modifiable alternate or express alternate PCB, you can override the default resource name at run time by setting a value for EZEDEST for a file or EZEDESTP for a printer in the program. EZEDEST is treated as a local variable. Setting EZEDEST for a record in one program does not affect EZEDEST in another program. An ADD function writes to the message queue identified by the setting of EZEDEST for that program. For more information on EZEDEST and EZEDESTP, refer to the *Programmer's Reference* document.

Message queue type

You can specify single-segment message queues (SMSGQ) or multiple-segment message queues (MMSGQ).

Single-segment message queues (SMSGQs)

For a single-segment message queue (SMSGQ), each record that is added to or scanned from the serial file is a complete message. The generated COBOL program issues an IMS PURG call between records that are added to a single-segment message queue. The generated COBOL program issues an IMS get unique for each SCAN I/O option.

Multiple-segment message queues (MMSGQs)

For multiple-segment message queues (MMSGQs), a series of adds to the serial file is treated as though each ADD function were for a segment of a single message. The message is not ended until you issue a CLOSE I/O option or reach a commit point. The generated COBOL program issues an IMS PURG call for the CLOSE I/O option. You can then begin adding segments of another message and close it. Multiple-segment message queues are not valid for printer files.

If you issue a SCAN I/O option for a MMSGQ serial file, the generated program issues an IMS get unique call to get the first segment of the message. Additional SCAN functions result in get next calls to get the remaining segments of the message. At the end of all the segments in a message, the generated COBOL program sets the NRF (no record found) record state. If you continue scanning, the generated program starts another series of get unique, followed by get next calls.

When no more messages are found, the generated program returns an EOF (end of file) state.

PCB number

You must also specify a PCB number for the serial file that is associated with a message queue. You must specify PCB 0 as the PCB number for a serial input file. PCB 0 is the number of the I/O PCB that is the only message queue used for input. If you use a serial input file, you must use a main batch or called batch program. The generated program handles all I/O PCB logic for main transaction programs.

You can specify the PCB number for a serial output file. The PCB number must be the number of a TP PCB in the PSB definition. The default PCB number is 1. You can only send output to PCB 0 using the CSPTDLI service routine. For more information about the service routines that you can use with IMS, refer to the *Programmer's Reference* document.

Defining records to use with message queues

When you define a serial record to associate with a message queue, you should define only the program data. The generated COBOL program adds the IMS message header (length, ZZ, and transaction code) for an ADD I/O option and remove it for a SCAN I/O option. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information.

Checking the results of serial file I/O options

When a serial file is associated with a message queue or GSAM database, the generated program issues a DL/I call to implement the I/O operation. When the DL/I call completes, VisualAge Generator Server for MVS, VSE, and VM performs the following functions:

- For SCAN I/O options, the record state is set based on the DL/I status code. The EZEUSR or EZEUSRID special function word is updated from the user ID field of the I/O PCB when the generated program issues a get unique call for the I/O PCB. This happens at the first SCAN for a serial file defined as a multiple-segment message queue (MMSGQ), and at each SCAN for a single-segment message queue (SMSGQ). The EZESEGTR special function word field is updated from the transaction name in the IMS message header after each SCAN that results in a get unique call for the I/O PCB.
- For an ADD or CLOSE I/O option, the record state is updated based on the DL/I status code.

After a DL/I call that involves either the message queue or GSAM, the EZEDL special function words are not updated. These EZEDL words are updated only for functions that access DL/I segment records. This allows a program written for a CICS transient data queue or an OS/VS serial file to

run consistently if the file is changed to a message queue or GSAM database in an IMS environment. If you need access to the PCB, you can use the EZEDLPCB special function word to move the PCB contents to a working storage record for direct access by your program. You should check the I/O error values to determine if `End Of File` was reached or an error occurred on the serial file. Refer to the *Programmer's Reference* document for more information on I/O error values.

Using printer files as message queues

For IMS/VS you must associate printer files with message queues. For IMS BMP, you can associate printer files with message queues. You associate printer files with message queues the same way you associate serial files with message queues, except only SMSGQ is valid for EZEPRINT. During IMS GEN, you must define the message queue name that you use in the runtime environment as a logical terminal. EZEDESTP can be used to change the printer destination at run time. For example, you could define a table of user IDs and the printer ID that each user normally uses. By setting EZEDESTP, you can route the printer output to a printer near the program user.

Using IMS functions from VisualAge Generator programs

Table 16 lists the IMS/VS functions that you can use in VisualAge Generator programs, as well as summarize how to use these functions.

Table 16. Comparison of IMS and VisualAge Generator Functions

IMS Function	VisualAge Generator Function	Comments
Type of Program		
Conversational	Main segmented transaction program with a PSB, a target system of IMS/VS, and a SPA	/SPA is a VisualAge Generator generation option.
Nonconversational (option 1) or message-driven fast path	Main segmented transaction program with a PSB, a target system of IMS/VS, and no SPA	Some VisualAge Generator functions cannot be used if the program is generated as message-driven fast path. See "Developing IMS fast path programs" on page 145 for more information.
Nonconversational (option 2) or message-driven fast path	Main single-segment transaction with a PSB and a target system of IMS/VS	Some VisualAge Generator functions cannot be used if the program is generated as message-driven fast path.

Table 16. Comparison of IMS and VisualAge Generator Functions (continued)

IMS Function	VisualAge Generator Function	Comments
Nonconversational or message-driven fast path without terminal maps	Main batch program with a PSB, that accesses the I/O PCB as a serial input file. In VisualAge Generator Server for MVS, VSE, and VM, reading messages using the I/O PCB results in a commit point.	Some VisualAge Generator functions cannot be used if the program is generated as message-driven fast path.
BMP	Main batch program with a PSB and a target system of IMS BMP	Transaction-oriented BMP uses message queue for input. Batch-oriented BMP does not use the message queue. VisualAge Generator treats a batch program that uses the SCAN I/O option for a file associated with the I/O PCB as a transaction-oriented BMP; other batch programs generated for the IMS BMP environment are treated like batch-oriented BMPs.
DL/I Batch	Main batch program with a PSB and a target system of MVS batch	
Normal MVS batch	Main batch program without a PSB and a target system of MVS batch	
Terminal and Printer Support		
Communication with the terminal	For segmented mode, a CONVERSE function for output and input or an XFER statement with a map for output and First Map for input. For single-segment mode, an XFER statement with a map for output and First Map for input.	
Dynamic printer support	EZEDESTP special function word set to alter print destination for DISPLAY	

Table 16. Comparison of IMS and VisualAge Generator Functions (continued)

IMS Function	VisualAge Generator Function	Comments
/FORMAT modname mapname	The IMS /FORMAT command can be used to display any map generated for IMS. The syntax of the command is /FORMAT modname mapname where modname is the map group name concatenated with O and mapname is the name of the map defined as the First Map for the program.	
Physical and logical paging	Not supported	Map definition does not support the definition of physical or logical pages.
SDF II to produce the MFS source	Not supported	Generation requires knowledge of the edit order and special control fields.
Database and File Access		
DL/I database definition and access	PSB definition, DL/I segment definition, and normal I/O options as provided for DL/I programs	VisualAge Generator creates default SSAs and sets the default PCB number.
DL/I fast path database definition and access	Same as DL/I database definition and access	DL/I call definition allows 2-position command codes. CSPTDLI can be used for FLD and POS calls.
DB2 database definition and access	SQL row definition and normal I/O options as provided for relational programs	
Message queue for secondary transactions	ADD and CLOSE I/O options to serial file defined as message queue. Alter destination with the EZEDEST special function word. Alternatively, use the CREATX service routine.	Use a batch program with a SCAN function to read the records inserted to the message queue by the previous program's ADD function or CREATX service routine.
GSAM	ADD, SCAN, and CLOSE I/O options to serial file defined as GSAM	
Message Switching		

Table 16. Comparison of IMS and VisualAge Generator Functions (continued)

IMS Function	VisualAge Generator Function	Comments
Immediate program-to-program message switch	An XFER statement with working storage or other record passed to new VisualAge Generator program or non-VisualAge Generator program	Requires the transaction name and PSB name to change because different load modules are involved and the PSB name must match the load module name. The map group name can change or stay the same. Can be used by either segmented or single-segment programs.
Deferred program-to-program message switch (Method 1)	First program ends with an XFER statement with a map name specified. Second program begins with First Map that matches the map displayed by the XFER statement.	Requires the transaction name and PSB name to change because different load modules are involved and the PSB name must match the load module name. Same map group must be used by both programs. Can be used by either segmented or single-segment programs.
Deferred program-to-program switch (Method 2)	CONVERSE with EZESEGTR set to next transaction name	Permits the transaction name change within the same VisualAge Generator program. The same PSB and map group must be used by both transactions. Can be used only by segmented programs.
Miscellaneous		
Basic checkpoint	CALL EZECOMIT	
Symbolic checkpoint	CALL CSPTDLI service routine	Refer to the *Programmer's Reference* document for more information.
Restart	CALL CSPTDLI service routine	Refer to the *Programmer's Reference* document for more information.
LOG call	CALL AUDIT service routine	Refer to the *Programmer's Reference* document for more information.

Table 16. Comparison of IMS and VisualAge Generator Functions (continued)

IMS Function	VisualAge Generator Function	Comments
PURG call	CLOSE I/O option for printer map or serial file associated with an output message queue	For printers and message queues
Asynchronous processing	CALL CREATX service routine	Refer to the *Programmer's Reference* document for more information.

Chapter 6. Developing CICS programs

This chapter describes design and development considerations for CICS for MVS/ESA, CICS for VSE/ESA, CICS for OS/2, CICS for AIX, CICS for Windows NT, and CICS for Solaris programs. You can perform the following functions using VisualAge Generator to develop CICS programs:

- Define programs for CICS
- Test CICS programs using the VisualAge Generator Developer test facility
- Generate COBOL or C++ programs to run in the CICS environment

The following sections in this manual contain additional information on developing programs for the CICS environment:

- "Accessing multiple DB2 plans in CICS for MVS/ESA" on page 133
- "DL/I considerations for the CICS environment" on page 108
- "Choosing between CALL and DXFR statements" on page 247
- "Choosing between XFER and DXFR statements" on page 248
- Chapter 4. Developing segmented programs

Refer to the VisualAge Generator documentation for the following information about CICS:

- Transferring Control in the CICS Environment in the *VisualAge Generator Client/Server Communications Guide* document
- Developing Client/Server Programs in the *VisualAge Generator Client/Server Communications Guide* document
- Administering on CICS for MVS/ESA in the *VisualAge Generator Server Guide for MVS, VSE, and VM* document
- Administering on CICS for VSE/ESA in the *VisualAge Generator Server Guide for MVS, VSE, and VM* document
- Preparing a generated program to run in CICS for MVS/ESA in the *VisualAge Generator Server Guide for MVS, VSE, and VM* document.
- Preparing a generated program to run in CICS for VSE/ESA in the *VisualAge Generator Server Guide for MVS, VSE, and VM* document.
- Preparing a generated program to run in CICS for OS/2 in the *VisualAge Generator Server Guide for Workstation Platforms* document.

Understanding CICS terminology

Transaction
: A unit of processing, consisting of one or more programs.

Task
: The processing of a transaction for a program user.

Conversational
The CICS term for running a program in nonsegmented mode. A conversational program consists of a sequence of alternating entries and responses between a user and the program. File and database position and locks, and storage resources are held across the terminal I/O operation.

Pseudoconversational
The CICS term for running a program in segmented mode. A pseudoconversational program consists of a series of single CICS tasks designed to appear to the user as a continuous conversation. File and database position and locks, and storage resources are released across the terminal I/O operation. The program must save conversation status before terminal output and restore it on terminal input.

Communication area (COMMAREA)
A data area used to transfer information between two programs within a transaction or between two transactions from the same terminal. When one program transfers to another, the COMMAREA can be any data area the transferring program can access. The transferred-from program can both pass data to that area and receive results in the area. The data area is usually the working storage area of that program.

Transaction Work Area (TWA)
A fixed length storage area allocated for each transaction task control area. Generated programs use a 1024 byte section of the TWA. The offset of the section of bytes is controlled by the /TWAOFF generation option.

Control Tables (CICS for MVS/ESA, CICS for VSE/ESA, and CICS for OS/2) or Resource Definitions (CICS for AIX, CICS for Windows NT, CICS for Solaris)
Definitions of resources used or managed by the CICS system. Each definition is created by using resource definition online (RDO) or by coding macros for the tables. The following is a list of the types of definitions:

Destination Control Table (DCT) or Transient Data Definition (TDD) Used to define transient data destinations for the system.

File Control Table (FCT) or File Definition (FD)
Used to define files used by the system.

Processing Program Table (PPT) or Program Definition (PD)
Contains information about each program. The VisualAge Generator Server for MVS, VSE, and VM programs, generated

COBOL programs, mapping services programs, map group format modules, and table programs must be defined in the PPT.

Program Control Table (PCT) or Transaction Definition (TD)
Defines the transaction identifiers that can be entered by program users. For each transaction, it also defines the related program that starts the processing for the transaction.

Resource Control Table (RCT)
Describes the interface between the CICS region and DB2, including the association of transaction codes to DB2 program plans. Information on defining this table is located in the manual on installing DB2 for your version of DB2.

Terminal Control Table (TCT) or Terminal Definition (WD)
Contains descriptions of terminals, their features, and operating information.

Temporary storage queue
A CICS managed file for storing intermediate results. Records in a temporary storage queue can be accessed serially or by a relative record number. Descriptions of the two types of temporary storage follow:

Auxiliary
A temporary storage queue that is stored on DASD. It can be recovered and maintained from one CICS run to the next.

Main A temporary storage queue that exists in the CICS address space. It is not recoverable and is not maintained from one CICS run to the next.

Transient data queue
A CICS managed file that is serially organized. Descriptions of the two types of transient data queues follow:

Extrapartition transient data queue
A CICS managed serial file in a system sequential data set or tape. The file can be an input file or an output file but not both. Extrapartition queues are not recoverable.

Intrapartition transient data queue
A transient data queue that is accessible to transactions running in a CICS region. The queue can be used for both input and output. On MVS, the queue is stored in a VSAM entry sequenced data set. Intrapartition queues can be recovered.

File techniques in CICS programs

Defining CICS programs is much the same as defining programs for other environments. There are some file technique considerations you should note that are specific to CICS.

Using temporary storage

In CICS, temporary storage is the primary method for storing data that must be available to multiple transactions. Data items in temporary storage are placed in queues with names assigned dynamically by the program storing the data. Temporary storage is implemented in two different ways: main temporary storage and auxiliary temporary storage. Main indicates that the queue is stored in main storage in space taken from the dynamic storage area. Auxiliary indicates that the queue is written to an entry-sequenced VSAM data set. CICS maintains an index of items in main storage.

Main and auxiliary storage have the following characteristics:

- Main temporary storage requires more virtual storage than does auxiliary. It should be used for small queues that have short lifetimes or are accessed frequently.
- Auxiliary temporary storage is designed for large amounts of data that must be stored for a long time or are accessed infrequently.
- Queues can be recovered in auxiliary temporary storage.

  Note that only one transaction at a time can update a recoverable temporary storage queue. Keep in mind the probability of enqueues as you design your program. You should also ensure that there are enough VSAM strings to eliminate as much contention as possible.
- If a task attempts to write to temporary storage and the space is not available, CICS suspends the task. The task is not resumed until another task frees the needed space in main storage or in the VSAM data set.
- VisualAge Generator Server for MVS, VSE, and VM use temporary storage to save information about the program during a segmented converse or to save a copy of the map during a transfer using an XFER statement with a map. You can use the /WORKDB generation option to specify whether the main or auxiliary temporary storage is to be used.

Accessing temporary storage from VisualAge Generator

A generated program running in the CICS environment can access CICS temporary storage as a serial or relative record. The following I/O options are valid when you access temporary storage:

- ADD
- SCAN
- UPDATE
- REPLACE
- DELETE

- INQUIRY
- CLOSE

The record file must have the file type specified as TEMPAUX (auxiliary storage file) or TEMPMAIN (main storage file) when the program is generated. The system resource name is the queue name associated with the temporary storage file.

Temporary storage files can be used by only one task at a time. The program enqueues (ENQ command with the NOSUSPEND option) on resource name EZETEMP-queuename when the queue is first accessed and dequeues (DEQ command) when the file is closed (CLOSE I/O option or end of program) or when recoverable resources are committed. Non-VisualAge Generator programs that access the same file should enqueue on the same system resource name while accessing the file.

Records in temporary storage have an additional byte added to the front of the record that indicates the status of the record:
X'01' indicates that the record has logically been deleted.
X'00' indicates that the record logically exists in the file.

The additional byte is added to the record definition and managed by VisualAge Generator Server for MVS, VSE, and VM. Do not include the additional byte in the VisualAge Generator record definition. However, if the temporary storage file is also used by a non-VisualAge Generator program, the program must allocate space for the byte, interpret the byte, and update it as VisualAge Generator does. Processing of the additional byte is as follows:

ADD or REPLACE I/O option
The byte is set to X'00'.

DELETE I/O option
The byte is set to X'01' and the record length is set to 1.

SCAN I/O option
Records with a value of X'01' are skipped.

INQUIRY or UPDATE I/O option
Records with a value of X'01' cause an NRF record state to be set.

The CLOSE I/O option does not delete temporary storage files. Use the EZEPURGE special function word to delete the file. The program enqueues (ENQ command with the NOSUSPEND option) on resource name EZETEMP-queuename when EZEPURGE is used and dequeues (DEQ command) after the queue is deleted.

Using transient data queues

Like temporary storage, intrapartition transient data consists of data queues in a single data set with an index in main storage. You can use transient data queues for many of the same purposes as an auxiliary temporary storage queue with the following differences:

- Transient data queue names must be defined in a DCT or TDD before CICS is started. Transient data queues do not have the same random access characteristics as temporary storage queues.
- Transient data queues must be read sequentially, and each item can only be read once. After a transaction reads an item, the item is removed from the queue and is not available to any other transaction.
- Items in a transient data queue cannot be changed.
- Transient data queues are always written to a data set.
- Writing items to a transient data queue can initiate a specific transaction when the trigger level for the queue is reached.
- A transient data queue can be physically or logically recoverable, and you can specify that you want areas of the entry sequenced data set (ESD) that have been written and read to be reused for new data.
- Because the commands for intrapartition and extrapartition data sets are the same, you can switch between the internal CICS facility and an external data set. You need only change the DCT or the TDD for the data set.

Accessing transient data queues from VisualAge Generator

A generated program running in the CICS environment can access a CICS transient data queue as a serial record using ADD and SCAN I/O options. The record file must have the file type specified as TRANSIENT when the program is generated. The system resource name is the DCT or TDD name for the queue as it is defined to CICS.

Using spool files in CICS for MVS/ESA

generated programs running in CICS for MVS/ESA can access JES SPOOL files if the serial or print file is associated with the SPOOL file type at generation.

The system resource name for a spool file depends on whether the file is an input or output file. The following format descriptions are for the input and output files:

Input file

Name in the format *userid.class*. The *userid* is a 4- to 8-character external writer name. CICS requires that the first 4 characters of the external writer name be the same as the first 4 characters of the CICS APPLID used to identify the CICS region to ACF/VTAM. *Userid* can be specified as an asterisk. *Class* is a 1-character spool class. *Class* is optional and defaults to "A." The maximum name size is 10 bytes.

Output file
Name in the format *nodeid.userid.class* where *nodeid* is a 1- to 8-character system node id, *userid* is a 1- to 8-character system user id and *class* is a 1-character spool class. *Userid* and *nodeid* can be specified using an asterisk. *Class* is optional and defaults to "A." If *class* is not specified, *userid* is also optional and defaults to the CICS user id (the same value as stored in EZEUSRID). The maximum name size is 19 bytes.

Refer to the CICS customization manual for more information.

Do not use spool files as temporary storage files that a program writes to and then reads. You can specify the same resource name for an output and input file, but the resource name represents a destination rather than a specific file. If you write to a spool destination and close the file, the file might not be immediately available as an input file from that destination and might be queued behind other files sent to the same destination.

For more information on spool file access in CICS, refer to the CICS customization manual.

SPOOL files are opened on first access and closed at program end, CLOSE I/O option for the file, or when recoverable resources are committed (EZECOMIT, EZEROLLB, end of transaction or segment).

The test facility does not support SPOOL file access.

Using spool files in CICS for VSE/ESA

generated programs running in VSE batch or CICS for VSE/ESA can create and write to a VSE/POWER queue member by associating a serial or print file as the SPOOL file type at generation. A serial or print output file associated as filetype SPOOL can be created and routed to the RDR, LST, or PUN VSE/POWER queue. generated programs running on CICS for VSE/ESA can also read from a VSE/POWER queue member by associating a serial file as the SPOOL file type at generation.

The first ADD to a SPOOL file creates a new VSE/POWER queue member and adds the data to the beginning of the file. Later ADDs place data following the previously added data until the file is closed. A CLOSE I/O option issued for a SPOOL file closes the VSE/POWER queue member.

Once a SPOOL file is closed, a later ADD function to that file creates a new VSE/POWER queue member. When adding to a SPOOL file that is to be routed to the LST VSE/POWER queue, you must be aware of the following: VSE/POWER LST queue members are opened by VisualAge Generator Server for MVS, VSE, and VM with the ASA option. This specifies that the report is

created using an American National Standard printer-control character at the beginning of each line of data. If the file is a serial file, you must ensure that valid carriage control characters are used. If the file is a print file, then VisualAge Generator Server for MVS, VSE, and VM adds the American National Standard printer-control characters for you.

SPOOL files are closed at program end, CLOSE I/O option for the file, or when recoverable resources are committed (EZECOMIT, EZEROLLB, end of transaction or segment). Any close indicates the end of the file.

Because users have a choice of a VSE/POWER queue destination when creating an output SPOOL file, they have the ability to create a file that is placed on the VSE/POWER RDR queue as a batch job. Note that a CLOSE I/O option issued against a SPOOL file that is a RDR queue member indicates the end of the file. A subsequent ADD to the RDR queue file creates a new RDR queue member to be processed as a separate batch job. Also note that when creating jobs, if a POWER EOJ statement is output, the POWER job is made available for running before the SPOOL is closed.

SPOOL System Resource Name Format

The system resource name format for an input SPOOL file (CICS for VSE/ESA only) is *userid.class*. The *userid* is a 4- to 8-character external writer name. CICS requires that the first 4 characters of the external writer name be the same as the first 4 characters of the CICS APPLID used to identify the CICS for VSE/ESA partition to ACF/VTAM. The user ID defaults to the contents of the EZEUSRID special function word if you specify the user ID as an asterisk (*). *Class* is a 1-character spool class. Class is optional and, if left blank, defaults to A. You cannot use an asterisk to default class for an input SPOOL file. The maximum name size is 10 bytes. The input spool file is read from the PUN VSE/POWER queue.

The following is the system resource name format for an output SPOOL file:

```
jobname.queue.class.disp.form.node.userid
```

with an additional parameter for CICS for VSE/ESA.

The additional parameter on CICS for VSE/ESA is **parm**, so the following is the format on CICS for VSE/ESA:

```
jobname.queue.class.disp.form.node.userid.fcb.copies.parm
```

The jobname parameter must be specified or explicitly defaulted by an asterisk (*). All other parameters can be defaulted by an asterisk (*) or by a blank. However, if a parameter is defaulted by a blank, all subsequent parameters also default.

jobname

1 to 8 characters that define the jobname for the VSE/POWER queue member. This option is used in all cases except on CICS for VSE/ESA and not using Report Control Facility (in effect, queue is PUN or LST). For these two cases, the value in jobname is ignored, and the VSE/POWER queue member jobname is the CICS for VSE/ESA program ID. For all other cases, an asterisk (*) in this field defaults to the VisualAge Generator file name for the record.

queue 3 characters that identify the destination VSE/POWER queue for the file:

- RDR for job output
- LST for list output
- PUN for punch output
- PRT for list output (using CICS Report Control Facility)

Any other characters for queue cause a spool name error. On CICS for VSE/ESA, an asterisk (*) or a blank in this field defaults to the PRT queue. On VSE batch, an asterisk (*) or a blank in this field defaults to the LST queue. On CICS for VSE/ESA, the CICS Report Control Facility (RCF) is used for files that specify RDR or PRT in this field. Basic CICS SPOOL support is used for files that specify PUN or LST in this field. Therefore, LST and PRT both specify that the file is to be a member of the VSE/POWER LST queue, but PRT uses RCF commands while LST does not. If you attempt to use RCF when you do not have RCF installed on your CICS system, CICS returns an error condition. This might be an AEY9 transaction abend, a NO SPOOL condition, or the message "SPOOLING SYSTEM IS NOT AVAILABLE". On VSE batch, the queue PRT can be used, but is changed to LST by VisualAge Generator Server for MVS, VSE, and VM.

When queue is PRT or LST, the file is opened by VisualAge Generator Server for MVS, VSE, and VM with the ASA option. This specifies that the report is created using an American National Standard printer-control character at the beginning of each line of data. If you are using a serial file, you must ensure that valid carriage control characters are used. If the file is a print file, then VisualAge Generator Server for MVS, VSE, and VM adds the American National Standard printer-control characters for you.

class A single character that specifies class. An asterisk (*) or blank in this field defaults to A.

disp A single character that specifies the VSE/POWER disposition status of the queue member once it is closed:

- D - process the job and delete it after processing

- H - hold in queue until released
- K - process the job and keep it in the queue after processing
- L - leave in queue until released

Any other characters for queue causes a spool name error. This field is not applicable when you are on CICS for VSE/ESA and not using the Report Control Facility (in effect, when you are on CICS for VSE/ESA and the queue is LST or PUN). An asterisk (*) or blank in this field defaults to 'D'.

form 4 characters that identify the form number for print output. An asterisk (*) or a blank in this field defaults to your location's standard form. This field is applicable when queue is LST in VSE batch or PRT in CICS for VSE/ESA and is ignored for all other queues.

node 1 to 8 characters that specify the system node ID. An asterisk (*) or a blank in this field defaults to the current system node ID.

userid 1 to 8 characters that specify the user ID. An asterisk (*) or a blank in this field defaults as follows: For CICS for VSE/ESA, if you are signed on to CICS (either through CSSN or VSE/ESA interactive interface sign-on panel) then userid defaults to the contents of EZEUSRID. If you are not signed on to CICS, then userid defaults to '*'. For VSE batch, userid defaults to 'ANY''. This default indicates that VSE/POWER might make the output available to any user that requests this output.

parm A string of characters to be used to specify output operands for files on the VSE/POWER LST queue. This option is used only if you are on CICS for VSE/ESA when the queue is LST, and is ignored for all other queues. This string is passed to CICS in the OUTDESCR option of the **VSE CICS SPOOLOPEN OUTPUT** command. Therefore, you must specify these characters in the correct format for the OUTDESCR option: the parameters use the same keywords and values as are used on the VSE/POWER LST statement for user defined output operands but the syntax varies slightly. For example if you want to use FORMDEF FORM1 and PAGEDEF PAGE1, the parm string is **'FORMDEF(FORM1) PAGEDEF(PAGE1)'** and the spool file might appear as follows:

```
JOBNAME1.LST.*.*.*.*.*.FORMDEF(FORM1) PAGEDEF(PAGE1)
```

VisualAge Generator Server for MVS, VSE, and VM handles calculating and inserting the length area required by CICS for VSE/ESA at the beginning of the string. The length of the parm string is variable and depends on the length of the spool file specification up to this point: the total length of the spool file specification cannot be over 65 characters.

fcbname
1 to 8 characters that specify the name of the form control buffer. This is the name of the FBC-image phase which VSE/POWER will use for printing the job output. The name phase must be cataloged in a sublibrary accessible from the VSE/POWER partition. If omitted, or if an asterisk (*) is specified, the system default FCB is used.

copies A number from 1 to 255 that specifies the number of copies to be printed. The default is 1.

Refer to the CICS customization manual for more information.

Do not use spool files as temporary storage files for a program that writes to a file and then reads the file. You can specify the same resource name for an output and input file, but the resource name represents a destination rather than a specific file. If you write to a spool destination and close the file, the file might not be immediately available as an input file from that destination and might be queued behind other files sent to the same destination.

For more information on spool file access in CICS, refer to the CICS customization manual.

SPOOL files are opened on first access and closed at program end, CLOSE I/O option for the file, or when recoverable resources are committed (EZECOMIT, EZEROLLB, end of transaction or segment).

The test facility does not support SPOOL file access.

Using VSAM files

The file type VSAM is used generically in CICS environments to refer to a serial, relative, or indexed file accessed via a CICS FCT or FD. The underlying file system might be called VSAM (on MVS or VSE, for example) or might have a different name (Shared File Server on CICS for AIX, for example). The VisualAge Generator record file name must be associated with a file with type VSAM when the program is tested or generated. The system resource name is the FCT name for the data set as it is defined to CICS.

For CICS, the CLOSE I/O option does not actually close the data set. CLOSE releases record locks and position in the file.

When accessing the same indexed data set using either two file names for the same physical data set or two file names that access a base data set and its alternate index, an indefinite deadlock can occur in CICS that does not raise the DED condition. When the file is not defined with LSRPOOLID equal NONE in the FCT and one function in a program has performed a SCAN on a file and another function requests an UPDATE or ADD to the same file (or alternate index) without ending the SCAN, this deadlock can occur. If you

design this type of file access into your programs, then ensure the LSRPOOLID is set to NONE in order to avoid the deadlock.

Using OS/2 files

A generated program running in the CICS for OS/2 environment can access OS/2 files through COBOL READ/WRITE statements. Serial, relative, and indexed organizations are supported. The record file must have the file type specified as OS2COBOL during generation. The system resource name is the OS/2 file name.

File sharing is not supported for COBOL managed files. Whenever a program opens a file, an exclusive lock is obtained for the file until it is closed.

Using AIX files

A generated program running in the CICS for AIX environment can access AIX files as serial files. The record file name must be associated with the file type SEQ during generation. The system resource name is the AIX file name.

File sharing is not supported for AIX files. Whenever a program opens a file, an exclusive lock is obtained for the file until it is closed.

Using Windows NT files

A generated program running in the CICS for Windows NT environment can access Windows NT files as serial files. The record file name must be associated with the file type SEQ during generation. The system resource name is the Windows NT file name.

File sharing is not supported for Windows NT files. Whenever a program opens a file, an exclusive lock is obtained for the file until it is closed.

Using Solaris files

A generated program running in the CICS for Solaris environment can access Solaris files as serial files. The record file name must be associated with the file type SEQ during generation. The system resource name is the Solaris file name.

File sharing is not supported for Solaris files. Whenever a program opens a file, an exclusive lock is obtained for the file until it is closed.

Using EZEDEST

The EZEDEST special function word can be used to dynamically change the physical file associated with a record at run time by having the program move the system resource name of the new file to **record-name.EZEDEST**. The new system resource is used on the next I/O option associated with the record file. If another resource is already open for the record, that file is closed before the new file is accessed. The new resource must have the same file type as the file type specified for the record file when the program was generated.

Printing techniques in CICS

Programs write printer data when the program processes a DISPLAY I/O option for a printer map. The printer output is in line character format with American National Standard printer-control characters. The printer data is written to a logical file named EZEPRINT. For CICS for MVS/ESA, you can associate the EZEPRINT file with either a transient data queue (type TRANSIENT) or a JES spool file (type SPOOL) at generation. For CICS for VSE/ESA, you can associate the EZEPRINT file with either a transient data queue (type TRANSIENT) or a VSE/POWER file (type SPOOL) at generation. For CICS for OS/2, you can associate the EZEPRINT file to a serial file (type OS2COBOL) at generation. For CICS for AIX, CICS for Windows NT, and CICS for Solaris, you can associate the file EZEPRINT to a transient data queue (type TRANSIENT) at generation.

Using transient data queues for printer output

If EZEPRINT is associated with a transient data queue at generation, the system resource name is the destination control table (DCT) name for the queue. You can define the destination for the queue as a system printer, a terminal printer, or a data set. If the destination is a terminal printer, you need to define a transaction that is started when data is written to the queue. The transaction runs the VisualAge Generator Server for MVS, VSE, and VM program FZETPRT. FZETPRT reads the queue and writes the data to the terminal printer identified in the DCT entry.

The program does not actually write the printer output to the transient data queue until the print file is closed. The printed output is accumulated in temporary storage. When the file is closed (CLOSE I/O option or end of transaction), VisualAge Generator Server for MVS, VSE, and VM enqueues on the transient data queue using the DCT name as the resource name, copies the printer output to the queue, and dequeues. The maximum number of print records that can be accumulated in the transient data queue is 32765. Write your program to close the print file before 32765 records are accumulated.

Using spool files for printer output on CICS for MVS/ESA

If EZEPRINT is associated with the SPOOL file at generation, the system resource name identifies the node, user or external writer identifier, and class that you want to spool the file. The name is in the format *nodeid.userid.class* where *nodeid* is a 1- to 8-character system node ID, *userid* is a 1- to 8-character system user ID, and *class* is a 1-character spool class. *Userid* and *nodeid* can be specified as an asterisk. *Class* is optional and defaults to "A". If *class* is not specified, *userid* is also optional and defaults to the CICS user id (the same value as stored in EZEUSRID). The maximum name size is 19 bytes.

Refer to the CICS customization manual for more information.

The spool file is opened as needed on DISPLAY I/O options and closed on CLOSE I/O options for a printer map, or when recoverable resources are committed (EZECOMIT, EZEROLLB, or end of transaction or segment).

The test facility does not support spool file access.

Using OS/2 files for printer output

Programs generated for the CICS for OS/2 environment use COBOL I/O (file type OS2COBOL) for print output. This means you do not need additional CICS entries as you do for CICS for MVS/ESA and CICS for VSE/ESA. The default system resource name is LPT1. LPT1 is usually your current printer. When you want to change the printer destination, you can move the new printer destination to EZEDESTP.

Using VSE/POWER files for printer output

Printing is initiated when a program processes a DISPLAY I/O option for a VisualAge Generator-defined printer map or printing can be initiated from a serial record defined as a SPOOL file when destination queue is the VSE/POWER LST queue.

When printing is initiated through the DISPLAY of a printer map, the printer output is routed to the file that is specified as the resource associated to EZEPRINT. The resource associated to EZEPRINT can be specified at generation or at run time (using the EZEDESTP special function word).

On CICS for VSE/ESA, if the resource associated to EZEPRINT is a SPOOL file, it is spooled to VSE/POWER. The user can specify that queue equal LST or PRT, causing the SPOOL file to become a VSE/POWER LST queue member. The user can specify the jobname, class, disp, form, node, and userid of the VSE/POWER LST queue member. These values are specified using the system resource name format for the spool file.

The CICS Report Controller can be used in conjunction with VisualAge Generator Server for MVS, VSE, and VM printer functions to provide ease of handling printed output.

Printing can also be initiated through serial records defined as filetype SPOOL on CICS for VSE/ESA. The system resource name format for the spool file enables program users to specify the VSE/POWER queue destination. If the queue equals LST or PRT, the file becomes a VSE/POWER LST queue member.

Using EZEDESTP

The EZEDESTP special function word can be used to dynamically change print file destination (transient data queue or spool file name) at run time by having the program move the new system resource name for the print file

before the DISPLAY option is run. The new resource must have the same file type as the file type specified for EZEPRINT when the program was generated.

Multiple print files can be open at the same time. A DISPLAY writes to the resource named in EZEDESTP at the time the DISPLAY option is run. A CLOSE for a printer map closes only the resource named in EZEDESTP. Any files not explicitly closed are closed at the end of the transaction or segment, or commit point for spool files.

The default value for EZEDESTP is the system resource name specified for the EZEPRINT file at generation. If you specified the generation option /PRINTDEST = TRMID and the program was started with a CREATX that had the `recip` parameter set to binary zeros and that specified a `prid`, then EZEDESTP is initialized to the value in `prid`. For OS/2, AIX, Windows NT, HP-UX, and Solaris, you can associate the file EZEPRINT to a serial file (type SEQ) in the resource association file that is used at runtime.

If the program was started by a non-VisualAge Generator program that specified the RTERMID on the START command, then EZEDESTP is initialized to the value specified for RTERMID.

Setting the recovery unit of work

The EZECOMIT service routine is used to indicate that the current recovery unit of work is complete and a new unit of work is to be started. This routine issues a CICS SYNCPOINT command.

There is an implicit EZECOMIT upon completion of a program (EZECLOS of a main program or an XFER statement). However, a CALL EZECOMIT can be issued at any time during execution of a program. It is advisable to issue a CALL EZECOMIT from within programs that have UPDATE, REPLACE, DELETE, or ADD with a logic loop at the completion of a logical unit of work so that multiple locks are not held by the program.

Special care must be taken to prevent deadlocks when data sets using VSAM Local Shared Resources (LSR) on CICS are accessed from a generated program. When using LSR and performing SCAN or SCANBACK processing on a file, you should issue a CALL EZECOMIT at the completion of a logical unit of work prior to attempting an operation requiring exclusive use of resources such as UPDATE, REPLACE, DELETE, or ADD. This releases the SCAN position on the data set allowing later updates to the data set.

If you are defining DXFR statements to replace previous XFER statements, note that the recovery unit of work holds across the transfer and you must define a CALL EZECOMIT if you want to end the unit of work.

A SYNCPOINT occurs on a DXFR statement under the following conditions:

- If a transfer to a non-VisualAge Generator program occurs and a PSB is scheduled
- If the /SYNCDXFR generation option is specified and a PSB is scheduled
- If /NOSYNCDXFR is specified for the transferring program, and if the transferring program had scheduled a PSB, and if different PSB names were identified during program specification for the two programs

On CICS for OS/2 systems, VisualAge Generator Server issues an SQL COMMIT WORK command before the CICS SYNCPOINT when SQL requests have been issued in the current unit of work.

Using CICS functions from VisualAge Generator programs

Table 17 lists the CICS functions that you can use in VisualAge Generator programs. The tables also summarize how to use these functions.

Table 17. CICS functions and how to represent them in VisualAge Generator

CICS Function	VisualAge Generator Function	Comments
Type of Program		
Conversational	Nonsegmented execution mode	
Pseudoconversational	Segmented execution mode	/WORKDB generation option specifies whether to use a main or auxiliary temporary storage queue for saving status across the terminal I/O.
Pseudoconversational, different transaction names	Segmented CONVERSE with EZESEGTR set to next transaction name	/WORKDB generation option specifies whether to use a main or auxiliary temporary storage queue for saving status across the terminal I/O.
Terminal and Printer Support		
Communication with the terminal	CONVERSE for input/output, DISPLAY for output, or an XFER statement with a map for output and First Map for input.	
System printer support	TRANSIENT file type for EZEPRINT file	Associate DCT for transient data queue with system printer
Terminal printer support	TRANSIENT file type for EZEPRINT file	Associate DCT for transient data queue with terminal printer and trigger FZETPRT transaction to write to printer.

Table 17. CICS functions and how to represent them in VisualAge Generator (continued)

CICS Function	VisualAge Generator Function	Comments
JES SPOOL file for printer output	SPOOL file type for EZEPRINT file	Resource name identifies node, spool writer or user identifier, and class
VSE/POWER SPOOL file for printer support	SPOOL file type for EZEPRINT file on VSE environment	Resource name identifies jobname, queue, class, disp, form, node, userid, and parm of the VSE/POWER LST queue member.
Dynamic printer support	EZEDESTP set to alter print destination for DISPLAY	
Database and File Support		
DL/I database definition and access	PSB definition, DL/I segment definition, and normal I/O options as provided for DL/I programs	VisualAge Generator creates default SSAs and sets the default PCB number.
PSB Scheduling	EZEDLPSB identifies PSB to be scheduled.	Scheduling is done automatically prior to the first DL/I operation in the unit of work.
PSB Termination	Done automatically on EZECOMIT, EZEROLLB, end of transaction or segment.	A SYNCPOINT occurs on a DXFR statement under the following conditions: If a transfer to a non-VisualAge Generator program occurs and a PSB is scheduled If /SYNCDXFR is specified and a PSB is scheduled If /NOSYNCDXFR is specified for the transferring program, the transferring program had scheduled a PSB and different PSB names were identified for the two programs during program specification.
Program restart following abnormal termination due to deadlock in queueing on database records	EZEDLRST	Switch indicating whether program was restarted
DB2 database definition and access	SQL row definition and normal I/O options as provided for relational programs	

Table 17. CICS functions and how to represent them in VisualAge Generator (continued)

CICS Function	VisualAge Generator Function	Comments
VSAM file support	VSAM file type for serial, relative, and indexed files	FCT or FD required for file; FCT or FD name is system resource name
Transient data queue support	TRANSIENT file type for serial files	DCT or TDD required for file; DCT or TDD name is system resource name
Function shipping for VSAM data sets and transient data queues	:FILELINK LINKTYPE=REMOTE in the linkage table for the file EZELOC special function word	
Specifying SYSID when function shipping.	:FILELINK LINKTYPE=REMOTE location=EZELOC in the linkage table entry for the file	
Main temporary storage queue support	TEMPMAIN file type for serial or relative file	Records have extra control byte in byte 1
Auxiliary temporary storage queue support	TEMPAUX file type for serial or relative file	Records have extra control byte in byte 1
JES SPOOL file support	SPOOL file type for serial file	Resource name identifies node (output only), spool writer or user identifier, and class
VSE/POWER SPOOL file	SPOOL file type for serial file	Resource name identifies spool writer or user identifier and class (input only); or jobname, queue, class, disp, form, node, userid, and parm for a VSE/POWER RDR, PUN, or LST queue member (output only).
Program Communications		
START transaction	An XFER statement without a map or CALL CREATX	
RETURN TRANSID	an XFER statement with a map or a segmented CONVERSE	
Function shipping for START transaction	CALL CREATX; :CRTXLINK LINKTYPE=REMOTE in linkage table entry for CREATX record	
Specifying SYSID when function shipping	:CRTXLINK LINKTYPE=REMOTE location=EZELOC in linkage table entry for CREATX record	
XCTL to a program	A DXFR statement	If a record is specified, it is transferred in the COMMAREA.

Table 17. CICS functions and how to represent them in VisualAge Generator (continued)

CICS Function	VisualAge Generator Function	Comments
XCTL to a program	A DXFR statement with NONCSP option	If a record is specified, it is transferred in the COMMAREA.
LINK to program with data in COMMAREA	CALL program; :calllink LINKTYPE=CICSLINK parmform=COMMDATA for called program in linkage table	
Distributed program LINK to program with data in COMMAREA	CALL program; :calllink LINKTYPE=REMOTE parmform=COMMDATA for called program in linkage table	
Specifying SYSID on distributed program LINK	:calllink LINKTYPE=REMOTE location= system name on linkage table entry for called program	
Specifying TRANSID on distributed program LINK	:calllink LINKTYPE=REMOTE serverid=transaction name on linkage table entry for called program	
Specifying SYNCONRETURN on distributed program LINK	:calllink LINKTYPE=REMOTE luwcontrol=SERVER on linkage table entry for called program	
Miscellaneous		
SYNCPOINT	CALL EZECOMIT	
SYNCPOINT ROLLBACK	CALL EZEROLLB	
JOURNAL call	CALL AUDIT	

Communicating between multiple CICS transactions

Programs running in a CICS environment can communicate with other programs in the same CICS region using shared tables. Tables defined as shared cause all programs in the same CICS region to use the same copy of the table until a new copy is requested.

In CICS environments, shared tables can be modified at run time. Because of this, multiple VisualAge Generator program transactions running in the same CICS region could use a shared table as a shared communications area.

This use of tables might have synchronization considerations depending on the specific CICS platform and the way the data is modified in the table.

- For CICS for MVS/ESA and CICS for VSE/ESA, modifications to shared tables are not synchronized across call statements or I/O options.
- For CICS for OS/2, modifications to shared tables are never synchronized. If synchronization is required across a call statement or an I/O option, or if it is required in the CICS for OS/2 environment, an external serialization method might be needed. For example, a non-VisualAge Generator program could be called to perform a CICS ENQ function while the table is being updated.
- For CICS for AIX, CICS for Windows NT, and CICS for Solaris, shared updateable tables are not supported.

Inter-transaction affinity considerations in a CICSplex

A *CICSplex* consists of two or more CICS regions that are linked using CICS intercommunication facilities. A CICS function called *dynamic transaction routing* supports load balancing by dynamically routing a transaction from a terminal to any of the regions that have the resources to process the transaction.

Inter-transaction affinity occurs when two or more CICS transactions pass information to one another in a way that requires the transactions to run in the same CICS region. When inter-transaction affinity exists, you must define the transactions to CICS so that they are routed to the same region.

The following topics describe transaction routing considerations for CICS programs generated using VisualAge Generator. For a more complete discussion of transaction routing refer to *Dynamic Routing in a CICSplex, SC33-1012*

Segmented programs

Segmented programs use a temporary storage queue (the *work database*) for saving the state of the program conversation across a CONVERSE or XFER with MAP function. To ensure that all segments of the conversation run in the same region, specify *Pseudo-conversation* affinity lifetime with *LUname* affinity for the affinity group to which the transaction belongs.

Sharing VisualAge Generator tables for update

Programs can update shared VisualAge Generator tables in the CICS environment. The shared table is stored in *getmained* CICS storage; any updates are accessible only to programs running in the same region. Any transactions dependent on passing information through a shared table must be routed to the same region.

Temporary storage queues

VisualAge Generator support for temporary storage queues requires that access to the queues be serialized. The generated program does this by using

CICS ENQ and DEQ with the queue name as the resource name. ENQ and DEQ are effective only within the scope of a single region.

To ensure that access to the queue is serialized, do one of the following:
- Define the temporary storage queue as a local queue.
- Route all transactions that access the queue to same region.
- Use a queue naming convention that includes the terminal ID from EZELTERM as part of the queue name so that a different queue is used for each terminal. Since only one transaction is active from any one terminal at a time, access to the queue is serialized.

Refer to the CICS manual for more information on using temporary storage queues with transaction routing.

Using a transient data queue for printed output

Printed output can be routed to a transient data queue. the program accumulates the printed output in a temporary storage queue. When the output is complete, the program copies the output to the transient data queue, using ENQ/DEQ to ensure that output from multiple transactions in the same system is not interspersed.

Because ENQ/DEQ are effective only within a region, define the queue as a local queue to prevent interspersed output from multiple regions.

Also if you've defined the queue to trigger the FZETPRT terminal printing program, define the transaction for FZETPRT as a local transaction in the region where the queue resides.

Error destination queue

Runtime error messages from VisualAge Generator host services can be directed to a transient data queue called the error destination queue. Define the queue as a local queue to each region in which a VisualAge Generator program can run to ensure that messages related to a single error are not interspersed with messages related to another error occurring at the same time in another region.

Disable on run unit failure

One of the options that can be specified using the diagnostic control utility is disabling a transaction whenever a run-unit error is detected for that transaction.

The disable action is implemented using the CICS SET function and is effective only for the region in which the error occurred.

CICS utility function region affinity

The VisualAge Generator Server for MVS, VSE, and VM CICS utilities perform functions that have region affinity; therefore, you must ensure that

the transaction is routed to the desired region based on user identifier, LU name, or alternate transaction name. Table 18 lists the utilities, default transaction identifier, and function description.

Table 18. Region Dependent Utilities

Utility	Default ID	Function Description
CICS Utilities Menu	ELAM	Menu for selecting the other utilities (except trace).
New Copy	ELAN	Loads new copy of program, map group, or table in region
Diagnostic Message Print	ELAU	Prints error message queue associated with the region.
Diagnostic Control Options	ELAC	Sets error reporting options for VisualAge Generator Server for MVS, VSE, and VM. Is region dependent if option file (FCT name ELACFIL) is defined as local to each region; is not region dependent if ELACFIL is defined as a shared file accessed through a file owning region.
Trace	ELAZ	Trace options set by this utility are saved in shared *getmained* storage in the region and are effective only in the region in which the ELAZ transaction ran.

Chapter 7. Developing programs for OS/400

This chapter describes the general considerations for developing programs targeted for the OS/400 environment.

Defining program native database files

The following sections contain considerations for preparing your VisualAge Generator programs and data for use in the OS/400 environment.

VisualAge Generator record organization-to-file conversion

VisualAge Generator programs use the OS/400 file system to implement VisualAge Generator I/O options for file I/O operations. The OS/400 file system consists of two types of files: *physical* and *logical*. Physical files contain the actual data stored on the system. Logical files present a based or reorganized view of data in a physical file. Table 19 shows the VisualAge Generator record organizations that support file I/O operations and the recommended OS/400 file types. You should select the VisualAge Generator record organization that best fits the program being developed. Refer to the *AS/400 Data Management* , SC41-3710 document for details on using physical and logical files.

Table 19. VisualAge Generator Record Organizations and Recommended OS/400 File Types

VisualAge Generator Record Organization	OS/400 File Type
Indexed	Physical file or logical file
Indexed alternate specification	Logical file
Relative	Physical file
Serial	Physical file

Creating and naming files

VisualAge Generator requires that all files accessed by the program exist on the system before using VisualAge Generator functions for I/O operations on the files. The file I/O functions do not create a file for the program user.

You specify the names of files used by a program as system resource names during generation. The default system resource name for a file is the name you assigned during record definition. You can change the default name to some other name. You can also explicitly qualify a file name by adding a library name, using the form *library/filename*. If you do not explicitly qualify a

file name, VisualAge Generator searches for the file in the library list (*LIBL) when the program runs on the OS/400 system.

To create a physical file, use the create physical file (CRTPF) command. You can create physical files having the relative and serial VisualAge Generator record organizations with or without data description specifications (DDS) information. Files with these organizations are *arrival sequenced* files. If you do not use the DDS source information to create these files, you must specify the record length for the file using the RCDLEN parameter on the CRTPF command.

Use the DDS source information provided by VisualAge Generator to create physical and logical files that have the indexed and indexed alternate specification VisualAge Generator record organizations. Use the create physical files CRTPF command to create physical files and the create logical file CRTLF command to create logical files. Files having these record organizations are considered *keyed sequenced* files. The DDS source information provides the means for specifying the key fields.

Note: You can use other methods for creating files, such as the Interactive Data Definition Utility and SQL. However, because DDS is the most commonly used method, it is used in this chapter.

Sharing database files

VisualAge Generator programs can share a file's record position or pointer with another program or program in the same job. Use the SHARE(*YES) parameter on the OS/400 CREATE, CHANGE, or OVERRIDE file commands for file sharing. SHARE(*NO) maintains the independence of a file's record position for each program accessing the file.

Relative record file initialization

Initialize file members that have the relative VisualAge Generator record organization before attempting to use them. Relative record files are represented on the OS/400 system by physical files whose records are retrieved and manipulated by a relative record number. When you update or add a record to a relative file member, a place must exist in the member for the record. For an update, that place must be a valid, existing record; for a new record, that place must be a deleted record. Use the Initialize Physical File Member (INZPFM) command to initialize records for relative file members.

Record lock considerations

The VisualAge Generator UPDATE I/O option reads a record from a file and locks the record as *exclusive allow read*. This means that no other user can update this record until the user who locked the record releases the record

with an I/O option other than UPDATE. If one user attempts to update a record that is already held for update by another user, a VisualAge Generator LOK error occurs.

Using data description specifications generated by VisualAge Generator

During generation, VisualAge Generator can generate DDS information from VisualAge Generator record definitions that are used for file I/O operations.

The DDS information generated by VisualAge Generator is useful only to an OS/400 system administrator or program developer. The system administrator can use the DDS source members, or modified versions of them, to create the files that do not already exist on the OS/400 system. Using the DDS source information to create the files qualifies these files for OS/400 data management functions, such as specifying key fields, unique keys, and logical files.

You are not required to use the DDS source information to create files because VisualAge Generator does not require that the files a program accesses be externally described. VisualAge Generator relies on the record definition, which is built into the *PGM object, for the structure of a record. However, using the DDS information guarantees agreement between the program's view of the record structure and the record data stored on the OS/400 system.

Restrictions on logical files

VisualAge Generator supports simple logical files that use only one record format. The DDS source information specifies only one file on the PFILE keyword. Refer to the *Data Description Specifications Reference, SC41-3712* document for complete details on the DDS keywords and their usages.

Considerations for native database commitment control

OS/400 Commitment Control Services enables more than one program to access the same data at the same time and prevents data inconsistency on the OS/400 system.

You can specify how a VisualAge Generator program uses OS/400 Commitment Control Services at the following times:

- Program run time
- Program development (within VisualAge Generator)
- Program generation, if you specify the /COMMIT option in the resource association file, the file is under commitment control when it is opened. The /NOCOMMIT option does not place the file under commitment control.

Program development considerations

During program development, you can specify commitment control explicitly in the program. However, be aware that VisualAge Generator Server for AS/400 performs some implicit commitment control when the program runs.

Explicit commitment control

While defining a program with VisualAge Generator Developer, you can specify explicit logic to issue commits and rollbacks for both OS/400 database files (relative record, sequential, and indexed) and SQL tables. This logic consists of calls to the special routines EZECOMIT and EZEROLLB. The developer can also set special function word EZECNVCM to the value 1, which causes changes to be committed when any CONVERSE function presents a map on a workstation.

Any explicit use of commitment control ends the current unit of recovery and begins another. OS/400 Commitment Control Services releases any record and file locks being held when changes are committed or rolled back.

Implicit commitment control

At certain times, VisualAge Generator Server for AS/400 automatically performs commitment control. This implicit commitment control is independent of any instances of explicit commitment control. As you design your programs, you can exploit implicit commitment control and avoid using explicit commitment control. Table 20 summarizes the implicit commitment control processing that VisualAge Generator Server for AS/400 does.

Table 20. Implicit COMMIT, ROLLBACK, and SQL Close Cursor Processing

Cause of Program Exit	Implicit Action	MAIN Program	CALLED Program
XFER	EZECOMIT	yes	N/A
	SQL Close Cursor	yes	N/A
DXFR	EZECOMIT	no	N/A
	SQL Close Cursor	yes	N/A
EZECLOS	EZECOMIT	yes	no
	SQL Close Cursor	yes	X
error	EZEROLLB	yes	no
	SQL Close Cursor	yes	X

Legend:
N/A Not applicable
X If the program started the run unit, yes.

Program runtime considerations

The following sections describe the considerations to keep in mind during program run time.

Starting and ending commitment control cycles

To use OS/400 Commitment Control Services, you must explicitly start and end a commitment control cycle using the start commitment control (STRCMTCTL) command to start the commitment control and the end commitment control (ENDCMTCTL) command to end the commitment control. VisualAge Generator Server for AS/400 does not implicitly start or end commitment control cycles. However, DB2/400 implicitly starts commitment control automatically for programs that use SQL I/O options. After commitment control is started for the job, both native database I/O and SQL I/O can use the common commitment control OS/400 provides.

If necessary, you can change the commitment control for an SQL program in the templates. The COMMIT parameter on the CRTSQLCBLI command sets the level of commitment control for SQL statements in a program. The LCKLVL parameter on the STRCMTCTL command does not affect programs that use only SQL I/O operations.

If no commitment control cycle is active and the program attempts to open a file requiring commitment control, the program ends with an error condition. Messages in the job log explain the exact nature of the error. The program ends abnormally under these conditions because it might attempt to explicitly commit changes to a file, but that is possible only with an active commitment control cycle.

Refer to the *AS/400 Data Management* , SC41-3710 and *AS/400 ILE COBOL/400 Programmer's Guide* SC09-1522 documents for more information on OS/400 commitment control services.

Considerations for using DB2/400 databases

This section describes the considerations to keep in mind when you use the DB2/400 database with VisualAge Generator. The following items require special considerations when you use DB2/400 with VisualAge Generator programs:

- DB2/400 DATE and TIMESTMP columns
- Recovery and database integrity support
- ANSI SQL support

Refer to the *VisualAge Generator Generation Guide* document for information on generating VisualAge Generator DB2 programs.

Using DATE, TIME, and TIMESTMP SQL columns

DATE and TIMESTMP columns in the database are associated with character (CHA) data items in record and map definitions. The values are in character format with separator characters when they are processed in a program. "Processed" means that either the value was received from the Database Manager product or that the value was set by a program statement or map edit. You code VisualAge Generator statements to set a date value using either a hard-coded, fixed format or the default date format. This section explains how the default format mask of a date value is established by VisualAge Generator Server for AS/400 and how to ensure that it matches the format mask expected or received by DB2/400.

Note: On other VisualAge Generator Server for MVS, VSE, and VM platforms, such as MVS, setting a date value is less of a concern because both VisualAge Generator Server for MVS, VSE, and VM and the Database Manager product extract the default date format dynamically during run time. On OS/400 however, DB2/400 establishes the date format at program precompile time whereas VisualAge Generator Server for AS/400 still establishes its default data format dynamically at run time. The two products must use the same date format for the SQL statements using DATE and TIMESTMP host variables to operate successfully.

The VisualAge Generator language date edit values SYSGREGRN and SYSJULIAN for map fields and the special function word EZEDTELC (long date in character format) produce date values using the VisualAge Generator default date format mask. VisualAge Generator Server for AS/400 establishes the default date format mask at run time by deriving it from the OS/400 job attributes of Date format and Date separator, when the first VisualAge Generator run unit for the job is started. You can control the default by manipulating the job attribute before the first VisualAge Generator run unit starts. (See OS/400 command CHGJOB, parameters DATFMT and DATSEP, for valid values.) VisualAge Generator Server for AS/400 expands the OS/400 2-digit year format to a 4-digit year format.

DB2/400 establishes the date and time format for each program during SQL precompile of the program. (See OS/400 command CRTSQLCBLI, parameters DATFMT and DATSEP, for valid values.) The default is the date format and separator value obtained from the attributes of the job in which the precompile is performed.

Recovery and database integrity considerations

VisualAge Generator programs can use all the recovery and data integrity features that DB2/400 provides. Most commitment control issues that affect DB2/400 programs are the same as those for native databases. This section discusses some differences.

DB2/400 databases are recoverable resources. If your program makes changes to a DB2/400 table and the program was prepared with commitment control (COMMIT parameter not equal to *NONE on the CRTSQLCBLI (Create SQL ILE COBOL object) command, the changes are committed to the database. The changes are committed to the database when the job or the activation group ends or when an implicit or explicit VisualAge Generator commit occurs. For more information on implicit and explicit commitment control, see "Considerations for native database commitment control" on page 191.

The CMTSCOPE (commit scope) parameter on the STRCMTCTL (start commitment control) command defines the scope of commitment control to be *JOB or *ACTGRP. DB2/400, when the first SQL statement runs, issues the STRCMTCTL command if it has not already been issued. Likewise, when the commit scope is complete, DB2/400 issues the ENDCMTCTL (end commitment control) command. If your program ends abnormally before the end of the logical unit work (LUW) or before an explicit VisualAge Generator rollback is requested, all changes that were made since the beginning of the LUW is backed out.

Commitment control for VisualAge Generator DB2/400 programs is controlled through the COMMIT parameter on the CRTSQLCBLI command. If *NONE is specified, the program does not run under commitment control. *NONE must be specified if the VisualAge Generator program issues the SQL DROP COLLECTION, GRANT, or REVOKE commands in an SQLEXEC I/O option. The default value for the COMMIT parameter in the VisualAge Generator DB2/400 preparation template (EFK24PSM.TPL) is *CHG. You can modify EFK24PSM.TPL so that a template symbolic parameter (sysparm) establishes the commit value during VisualAge Generator program generation. This template modification enables easy variations of the COMMIT value among multiple programs. Refer to the *VisualAge Generator Generation Guide* document for information on modifying templates. Commitment control using DB2/400 is no different from commitment control using native database files except as noted here.

ANSI SQL support

Before running a program, the SQL statements need to be prepared and analyzed. DB2/400 can analyze SQL statements to verify that they meet the ANSI SQL standards. To use ANSI SQL in a VisualAge Generator DB2/400 program, specify the /ANSISQL generation option so that any SQL statements in the program are generated to the ANSI SQL format. If you use ANSI SQL, tailor the DB2/400 VisualAge Generator template (EFK24PSM) so that the FLAGSTD parameter on the CRTSQLCBLI command is *ANS.

Compatibility considerations

This section describes compatibility considerations for preparing programs to run in an OS/400 environment.

CALL statement error handling

You can receive the escape message sent by a CALLED program by using the REPLY option on the CALL statement. The escape message ID is placed in the special function word EZERT8. The escape message ID is 7 alphanumeric characters, such as the following:

```
aaannnn
```

Where *aaa* is usually a message ID prefix, which are alphabetic characters and *nnnn* is a character value consisting of hexadecimal digits in the range of 0 through 9 and A through F.

Calls to server programs follow the same EZERT8 values as on other server program platforms.

Variable length records

VisualAge Generator does not support variable length records for OS/400.

DBCS data type

VisualAge Generator supports the DBCS data type. The MIX data type requires program-programmer management of shift-out and shift-in DBCS data delimiters when setting or referencing values of MIX data items.

DBCS customers using RISC hardware can use OS/400 Version 3.6 or later. DBCS customers using non-RISC hardware must use OS/400 Version 3.2 or later.

Message tables

VisualAge Generator message tables are implemented as OS/400 message files. Enter GO CMDMSGF from any OS/400 system command line for a menu of message file commands.

As a result of the message table to message file conversion, VisualAge Generator programs generated for OS/400 cannot reference message tables using VisualAge Generator statements and expressions that operate on table row data. Examples of such statements are MOVE, FIND, RETR, conditional expressions, and assignment operands or receivers.

Serial file I/O

Serial files in VisualAge Generator-generated programs are opened for write access in one of two methods (OUTPUT or EXTEND), which affect the disposition of the existing file. OUTPUT and EXTEND are COBOL OPEN verb

phrases. The phrase produced in the OPEN statement for a particular file depends on the VisualAge Generator resource association file type value at generation time.

VisualAge Generator supports two file type values, VSAM and SEQ. For serial files, you can use either VSAM or SEQ file types. The VSAM file type produces the EXTEND phrase. The EXTEND phrase writes append records to the end of the existing file. The SEQ file type produces the OUTPUT phrase. The OUTPUT phrase clears the file and writing begins at the first record position.

OS/400 and COBOL/400 attempt to manage conflicting open methods when a file is already open in the job, taking into consideration its SHARE status. Refer to the *ILE COBOL/400 Programmer's Guide* , SC09-1522 document and the COBOL runtime messages for more information on serial files.

OS/400 file attribute SHARE

The following table depicts program behavior for specific and general file operations depending on the OS/400 file SHARE attribute.

Table 21. Program Behavior for the SHARE Attribute File Operations

Subject	SHARE(*YES)	SHARE(*NO)
Record position within a file.	Record position is shared between programs.	Record position is independent between programs.
CLOSE I/O option. (implemented as COBOL CLOSE statement)	Virtual or absolute depending on whether the file was already open when the program was started.	Absolute CLOSE of a file. Record position is lost. (Poor performer)
First use of a VisualAge Generator file I/O option on each file. (implemented in part as COBOL OPEN statement with INPUT, OUTPUT, or INPUT-OUTPUT phrases, depending on use of other VisualAge Generator I/O Options for same file in the program)	• Must be done in each program that uses the file. • Run time hard error if the OPEN phrase is not compatible with the file open type of the program that actually opened the file. (If EZEFEC is 1, then program logic will handle the error).	• Must be done in each program that uses the file. • File OPEN phrases are independent in each program, so they do not have to conform to each other.

Table 21. Program Behavior for the SHARE Attribute File Operations (continued)

Subject	SHARE(*YES)	SHARE(*NO)
EZECLOS or anytime a program ends.	• Main VisualAge Generator run unit program. All files in the run unit are CLOSED, but the effect is virtual or absolute depending on whether the file was already open when the program was started. • Subordinate VisualAge Generator run unit program. File is kept OPEN, and the record position is saved, if the program is restarted in the current VisualAge Generator run unit.	• Main VisualAge Generator run unit program. All files in the run unit are CLOSED, and the effect is absolute. • Subordinate VisualAge Generator run unit program. File is kept OPEN, and the record position is saved, if the program is restarted in the current VisualAge Generator run unit.

File I/O status in EZERT8

EZERT8 contains the file I/O status code. Use the /SYSCODES generation option to control the codes that are returned for file I/O errors. The /SYSCODES generation option value does not affect the use of VisualAge Generator mnemonics.

- If /NOSYSCODES is specified, EZERT8 contains CSP/AE system *independent* codes.
- If /SYSCODES is specified, EZERT8 contains system *dependent* access method return codes. For the VisualAge Generator COBOL/400 system, that is the COBOL file status key value. Refer to the *ILE COBOL/400 Reference* , SC09-1523 document for information on the COBOL File Status Key values. Refer to the VisualAge Generator Developer online help system for details on the format of EZERT8.
- By contrast, EZERT8 on CSP/AE unconditionally contains the message ID sent from the I/O routine that detected the OS/400 file I/O error.

The following table shows the correspondence between status key values, mnemonics and EZERT8. There is a many-to-1 correspondence between the file status key values and EZERT8.

Table 22. Correspondence between Status Key Values, Mnemonics, and EZERT8

EZERT8 - /SYSCODES, COBOL File Status Key values	VisualAge Generator Mnemonics	EZERT8 - /NOSYSCODES
00,05,07	NORMAL	000
02	DUP, ERR	103
04 (Var record format)	NORMAL	000
04	FMT, ERR, HRD	220
10,14,46	EOF, ERR	102
22	UNQ, ERR	206
23 (START)	EOF, ERR	102
23	NRF, ERR	205
24,34 (access method not relative or relative key not 0)	FUL, ERR, HRD	25A
35	FNF, ERR, HRD	251
38	FNA, ERR, HRD	218
39,95	FMT, ERR	220
9D	LOK, ERR, HRD	381

For all other file status codes, EZERT8 is set based on the type of request as shown in the following table:

Type of Request	VisualAge Generator Mnemonics	EZERT8 - /NOSYSCODES
OPEN	ERR, HRD	500
CLOSE, UNLOCK	ERR, HRD	989
READ, START	ERR, HRD	987
WRITE	ERR, HRD	988

UNQ and DUP I/O error mnemonic enablement

The use of either UNQ or the DUP file I/O error mnemonic requires special action to ensure predictable results. Ignoring this action and its consideration causes the test of the expression using these mnemonics to yield a result that is inconsistent with the actual file I/O error status.

For VisualAge Generator to enable detection of a duplicate record key when writing to an indexed file, the file (or its associated physical file if the file is

logical) must be defined with the OS/400 data description specification (DDS) language with the UNIQUE DDS file level keyword.

To enable detection of duplicate record keys when reading an indexed file, use a VisualAge Generator resource association file during program generation. Refer to the *VisualAge Generator Generation Guide* document for more information on the resource association file and its contents.

Considerations for VisualAge Generator map definition and runtime behavior

This section contains considerations for map definition for the OS/400 environment. On 5250 workstation devices, the limitations of a map are based on the minimum control unit capability. VisualAge Generator issues a warning if you generate a map that exceeds the capability of a minimum control unit.

Maps displayed on 5250 devices

The following compatibility considerations apply for maps displayed on devices, including double-byte character set (DBCS) devices, in the 5250 workstation family.

- Row 1 column 1 on the screen cannot contain data. It must be blank or contain a field attribute byte. Maps generated for the 5250 workstation cannot use row 1 column 1 as part of an input field.
- The number of variable fields allowed on the screen varies with the control unit to which the display device is attached. The limitation is a maximum of 256 input fields for 5250 control units.
- The following device types for maps are recommended for the OS/400 environment. Other device types that are 27 x 132 or less dimensionally are compatible. The device type that best fits the physical device is used. When a map group is defined for the OS/400 environment, include the appropriate choices from this list during device selection.

Device type	Characteristics
ANY-2D	For 24 × 80 screens
ANY-5D	For 27 × 132 screens
PRINTER	For single-byte printer maps
5550D	For maps containing DBCS variable fields
5550P	For DBCS printer maps

Maps containing DBCS fields

The following compatibility considerations apply to maps containing DBCS fields:

- OS/400 DBCS workstations do not display double-byte characters that start in column 80. Instead, a single-byte X is displayed in column 80 and in column 1 of the next line. To avoid this effect, do not define double-byte fields that span lines.
- When field outlining is specified for a field on a map, no data other than blanks can be displayed in the first 3 bytes on the map (row 1 columns 1 through 3). In addition, row 1 column 4 can only be a blank or an attribute byte.

5250 family keyboard considerations

The following table shows the mapping used for 5250 workstation keys when a VisualAge Generator program runs. All other keys (cursor movement, Enter, and Reset) have the same effect as in System/370 environments.

Table 23. VisualAge Generator Functions on OS/400 5250 Workstation

OS/400 5250 Workstation Key	VisualAge Generator Function
Help (operator error mode)	Displays Help for map
Help (not operator error mode)	Displays 'Help Not Available'
Print	Prints screen to local printer
Attn	Trace function or active attention handler
Clear	Clears screen
Rec Backspace	Clears screen
Field Exit (Newline with Erase EOF)	Sets modified data tag (MDT)
Roll Up or Page Down[1]	VisualAge Generator PA1 definition
Roll Down or Page Up[1]	VisualAge Generator PA2 definition
F1 through F12	VisualAge Generator PF1 through PF12
F13 through F24	VisualAge Generator PF13 through PF24

1: You can reverse the meaning of the roll keys in your online user profile. If you do reverse the meaning of the roll keys, then Roll Down becomes Page Down, and Roll Up becomes Page Up, which is the opposite of the entries in the table.

Print maps and spooled output

In VisualAge Generator, a printer map is intended for printer output. The printer map represents the physical layout of associated printed output for file name EZEPRINT.

The VisualAge Generator Server for AS/400 printer file QVGNPRNT formats the output of VisualAge Generator printer maps. The default printer file QVGNPRNT is shipped with VisualAge Generator Server for AS/400.

The default characteristics for the QVGNPRNT printer file are as follows:

Length
Lines per page is 66

Width Positions per line is 132

Control character
*FCFC

Refer to the *VisualAge Generator Generation Guide* for information on resource association files.

Performance considerations

This section describes considerations that affect the performance of VisualAge Generator programs at run time. This section familiarizes you with the attributes and concepts of various VisualAge Generator program implementations so you can apply general OS/400 recommendations that address performance tuning.

OS/400 system and program performance tuning is no different for VisualAge Generator programs and related objects than for other objects of the same type on OS/400. Refer to the *AS/400 LPS: Performance Tools/400* , GC41-3055 document for more information on OS/400 system tuning tasks and issues.

VisualAge Generator program linkage on CALL statements

The default VisualAge Generator program linkage for targets of a CALL statement is DYNAMIC. This means that the COBOL statement produced to affect the CALL is a CALL *identifier* type of statement. This is the opposite of a CALL *literal* type of statement, which associates a STATIC program linkage type in VisualAge Generator. For the best runtime performance on a CALL statement, use the VisualAge Generator STATIC program linkage type with COBOL/400 because it resolves to the target program object, sets a system pointer, and continues to use that system pointer throughout the duration of the COBOL run unit.

The VisualAge Generator DYNAMIC program linkage type is implemented so that for each CALL statement, COBOL/400 resolves to the target program object. To eliminate the need to constantly resolve objects, use the VisualAge Generator STATIC program linkage type. Do this at generation time by naming a linkage table file (/LINKAGE=) and by using the calllink tag within the file so that the CALL target is handled as specified by the STATIC program linkage type.

An example of a linkage table file that affects all program targets named in generated programs by changing the VisualAge Generator default from DYNAMIC to STATIC is as follows:

```
:CALLLINK applname=* linktype=STATIC.
```

Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on creating and using linkage table files.

Loading tables and map groups

How you load tables and map groups can affect the performance of your VisualAge Generator programs. You can improve the performance for loading tables and map groups by storing the binary tables and map groups in Integrated File System (IFS) stream files rather than in database files: QVGNTAB (for tables) and QVGNMAPG (for map groups).

Use the CPYTOSTMF (Copy to Stream File) command to copy the binary table and map group from the database file to the stream file. The stream file must reside in the /QVGN subdirectory in the root file system; otherwise, VisualAge Generator Server for AS/400 cannot find the table and map group as stream files. The /QVGN subdirectory is automatically created at installation.

It is recommended that you only copy VisualAge Generator tables and map groups to the IFS stream files after the VisualAge Generator program system is placed into production. Programs must be in production because the subdirectories in the IFS stream files, in this case /QVGN (which contains the tables and map groups), are scoped to all OS/400 jobs on the system. Testing of programs that use tables and map groups stored in the IFS stream files always access the stream file instance of the tables and map groups rather than the developer copy of the tables and map groups in database files *LIBL/QVGNTAB and *LIBL/QVGNMAPG.

Refer to the *AS/400 Integrated File System Introduction* SC41-3711 document for more information on the integrated file system. IFS system commands can be found by typing GO DATA on an OS/400 command line.

Note: All binary image table and map group files that you want copied to stream files reside in the same root subdirectory /QVGN. Ensure that all tables and map groups have unique names. Tables and map groups that occur within the ENVY package/application have unique names, but table and map group names from various library sources might collide.

If you do not copy tables and map groups into stream files in the /QVGN root subdirectory, loading tables and map groups can be slower. The binary table and map group files are stored in the native library system as database files in QVGNTAB and QVGNMAPG. This occurs automatically during preparation.

You can use any combination of tables and map groups in stream files and database files. To load tables and map groups, VisualAge Generator Server for

AS/400 first searches the /QVGN subdirectory for the table or map group. If the table or map group does not exist as a stream file, VisualAge Generator Server for AS/400 loads the table and map group from the *LIBL/QVGNTAB and *LIB/QVGNMAPG database files.

Using positive sign values for PACK and NUM Data types

VisualAge Generator has two sets of "packed" and "numeric" data types. The difference between the sets is the positive sign values of F or C.

Program developers can control the ILE COBOL/400 positive sign value stored in packed and numeric data items via the OS/400 option /POSSIGN=F or /POSSIGN=C. ILE COBOL sets the positive sign value. VisualAge Generator switches a positive sign value from F to C or from C to F, depending on the data item type and /POSSIGN= generation values. To switch a sign value, VisualAge Generator programs perform an ILE procedure call to VisualAge Generator Server for AS/400. To minimize the calls to VisualAge Generator Server for AS/400 and improve runtime performance, generate your programs so that the most used data types receive an ILE COBOL/400 sign value that is right for the VisualAge Generator positive sign.

The following table shows the affect of the /POSSIGN= generation value on ILE COBOL/400 positive sign values, VisualAge Generator data types, and VisualAge Generator sign values:

VisualAge Generator Data Type	VisualAge Generator Positive Sign Value	ILE COBOL/400 Positive Sign	ILE COBOL/400 Data Type
PACK	C	/POSSIGN= value	S9 COMP-3
NUMC	C	/POSSIGN= value	S9 USAGE DISPLAY
PACF	F	/POSSIGN= value	S9 COMP-3
NUM	F	/POSSIGN= value	S9 USAGE DISPLAY

Security considerations

This section describes the considerations for controlling access to programs and administration activities. Object security for VisualAge Generator programs and related objects on OS/400 is no different from security for any other objects on the OS/400 environment.

Use the standard object authority management commands on OS/400 to manage access to VisualAge Generator programs and related parts. These commands control user access to programs and activities. The commands include the following:

GRTOBJAUT
: To grant the user object authority

RVKOBJAUT
: To revoke the user's object authority

EDTOBJAUT
: To edit the user's object authority

CHGOBJOWN
: To change the object's owner

Refer to the *AS/400 Security - Basic* SC41-3301 document for more information on system and object access management.

Chapter 8. Allocating and associating files

Whenever you generate a program that uses serial, relative, indexed, or print files, you may specify the file type and a system resource name, using the resource association file at generation time. For serial files of type SEQ on VSE batch, you must also supply a system resource number. The file type identifies the access method used to read or write the file. The system resource name associates the file defined in the VisualAge Generator program with a physical file, queue, or data set in the target environment.

For some files, the generated program can dynamically allocate the physical file at run time, or dynamically change the system resource name used for file association.

Dynamically allocating files

The generated program dynamically allocates existing files for the following types of files in the following environments. When dynamic allocation is used, the system resource name is the name of the data set or file to be allocated.

- VSAM (file type VSAMRS) or sequential (file type SEQRS) files on MVS/TSO, MVS batch, IMS BMP, VM CMS, or VM batch. The system resource name must be specified as an MVS data set name, or on VM CMS, a VSE data set name, or a CMS file name. For serial and print files, the data set name can include a partitioned data set member name in parentheses.

  The program checks for a preexisting file allocation using the VisualAge Generator file name as the DD name before attempting a dynamic allocation on an MVS or VM system. Also, if the system resource name is less than 9 characters (less than 8 characters for VSAM on VM CMS) and does not contain any separators (periods or parentheses), it is assumed to be a DD name instead of a data set name, and the program attempts to access a file pre-allocated using that DD name before it attempts a dynamic allocation with a data set name.

  If dynamic allocation is used, the data set is allocated with the disposition of SHR. Therefore, when the data set is opened for output, the default disposition of OLD is used.
- IBM VisualAge for COBOL for OS/2 files (file type OS2COBOL) on CICS for OS/2. The system resource name is the OS/2 file name including drive and path information.
- ILE COBOL/400 files (filetype SEQ). If the file is not found at the time of open, the system creates the file.

- Sequential files (file type SEQ) on OS/2, AIX, CICS for AIX, Windows NT, CICS for Windows NT, HP-UX, Solaris and CICS for Solaris. The system resource name is the system file name including path information.

Note: Dynamic allocation of files is not supported in CICS for MVS/ESA and VSE environments.

Dynamic allocation is not successful on the first I/O function that accesses the file and are treated as hard I/O errors. If the program is handling hard I/O errors (EZEFEC is set to 1 and an error routine is specified), the program can test the record status indicators (FNF, file not found, or FNA, file not available) to determine whether the file could not be accessed because the file did not exist or was in use by another program.

Dynamically associating files

The system resource name associated with a file can be dynamically modified during program run time for most types of files using the EZEDEST and EZEDESTP special function words. For remote CICS files, the system where the file is accessed can be dynamically modified using the EZELOC special function word.

Invoking an I/O function using EZEDEST

The EZEDEST special function word must be qualified with a record name. Whenever an I/O option is performed for a record, the program performs the I/O operation on the system resource associated with the record file. You can associate the resource with the file using the following methods:

- The program moves a value to EZEDEST special function word.

  This method overrides the association specified using either of the other methods.
- The program user preallocates the record file name with a DD statement, with an ALLOC command (only with file types SEQ, SEQRS, VSAM (non-CICS), and VSAMRS only), or with a VM FILEDEF or DLBL command.

  This method overrides the next method.
- You specify a system resource name for the record file during generation or test.

The previously opened physical file is closed when an I/O option for a record with that file name is processed and the EZEDEST special function word is modified.

Invoking an I/O function using EZEDESTP

The EZEDESTP special function word contains the resource name of the print file. Whenever a function is invoked to print a file, the program prints to the

system resource associated with EZEPRINT. You can associate the resource with the file using the following methods:

- The program moves a value to EZEDESTP.

  This method overrides the association specified using either of the other methods.
- The program user preallocates EZEPRINT (EZEPRIN for VSE systems) to a system resource name (file types SEQ and SEQRS only). On VM CMS, this is done with a FILEDEF command. If the file is to go directly to the virtual print queue (as opposed to CMS disk space), use RECFM VA and LRECL 133 on the FILEDEF command. That is, use the following command:

  ```
  FILEDEF EZEPRINT PRINTER (RECFM VA LRECL 133
  ```

  To create the file on disk, enter the following command:

  ```
  FILEDEF EZEPRINT DISK fname ftype fmode (RECFM V LRECL 133
  ```

  This creates a file on disk with the carriage control characters included. The CMS PRINT command with the CC option can then be used to print the file. This method overrides the last method.
- You specify a system resource name for EZEPRINT during generation or test. For OS/2, AIX, CICS for AIX, Windows NT, CICS for Windows NT, HP-UX, Solaris, and CICS for Solaris, you specify EZEPRINT at runtime in the resource association file.

For some types of files, dynamic file association also allows multiple physical printer files to be open simultaneously.

If multiple printer files are supported, previously opened files are not closed when a printer map is displayed and EZEDESTP is modified. Position is maintained for each open file. A CLOSE I/O option is effective only for the file currently named in EZEDESTP. If multiple printer files are not supported, the previously opened file is closed when a printer map is displayed and EZEDESTP has been modified.

Supported file types

Table 24 shows the file types that can be used with EZEDEST and EZEDESTP. The table also indicates whether multiple files can be open simultaneously for that file type.

Table 24. File Types Supported By EZEDEST and EZEDESTP

File Type	EZEDEST	EZEDESTP	Multiple Print Files Open
GSAM	No	No	No
MMSGQ	Yes	No	No
OS2COBOL	Yes	No	No

Table 24. File Types Supported By EZEDEST and EZEDESTP (continued)

File Type	EZEDEST	EZEDESTP	Multiple Print Files Open
SEQ (OS/400)	Yes	Yes	No
SEQ (AIX, OS/2, Windows NT, HP-UX, Solaris, CICS for AIX, CICS for Windows NT, CICS for Solaris)	Yes	Yes	No
SEQ (Other)	No	No	No
SEQRS	Yes	Yes	Yes
SMSGQ	Yes	Yes	No
SPOOL (AIX, HP-UX, OS/400, Solaris)	No	Yes	No
SPOOL input file (Other)	Yes	No	No
SPOOL output file	Yes	Yes	Yes
TEMPAUX	Yes	No	No
TEMPMAIN	Yes	No	No
TRANSIENT	Yes	Yes	Yes
VSAM (non-CICS) (non-OS/400)	No	No	No
VSAM (CICS)	Yes	No	No
VSAMRS	Yes	No	No
VSAM (OS/400)	Yes	No	No

Sharing an MVS or VSE VSAM data set in a run unit

To share a VSAM data set, you can associate a VSAM data set with more than one file name in a run unit.

When multiple file names are associated with a single VSAM data set, some considerations depend on the file type specified when the program is generated. These considerations also apply when accessing an indexed data set using more than one index.

The following considerations are based on the generation file type:

VSAMRS, VSAM on MVS CICS or VSE CICS

I/O operations exhibit the behavior expected when specifying the DSN parameter on the MACRF option of the GENCB. The VSAM share options control whether sharing is permitted between run units, but not within a single run unit.

VSAM

COBOL I/O operations exhibit the behavior expected when specifying the DDN parameter on the MACRF option of the GENCB. The VSAM share options control whether the operation is permitted.

When more than one run unit attempts to access a data set, you must consider the VSAM share options specified for the data set.

Chapter 9. Developing programs containing DBCS

This chapter describes VisualAge Generator design considerations for programs that contain double-byte character set (DBCS) and mixed data (MIX).

Note: In this chapter, references to *host* means System/370 (MVS, VSE, and VM) and OS/400 environments. References to *workstation* means OS/2, AIX, Windows, HP-UX, and Solaris environments.

In VisualAge Generator, a DBCS data type contains only double-byte characters, where a single character is represented by 2 bytes of data. A MIX data type allows the presence of DBCS data in the same field as single-byte character set (SBCS) data.

VisualAge Generator allows certain part names to be DBCS names. DBCS and mixed data fields are also supported for map fields, record fields, table columns, and literals.

DBCS field outlining is supported for VisualAge Generator maps. Field outlining enables horizontal and vertical bars to be drawn around a field on a map. Any combination of vertical and horizontal lines is supported.

DBCS support for host environments and workstation environments differ:

- Host environments support the EBCDIC DBCS code set.

  In mixed single-byte character set/DBCS fields, the EBCDIC character sets require special delimiter characters to identify DBCS characters. A single-byte, shift-out (SO) character indicates that the data that follows is DBCS. A single-byte, shift-in (SI) character indicates that the data that follows is SBCS.
- Workstation environments support the ASCII DBCS code set.

  The ASCII character sets do not require the use of shift-out/shift-in (SOSI) codes to delimit DBCS characters. Instead, a range of code points is set aside to be the first byte of a DBCS character. These code points have no meaning as single-byte codes. In addition, the data stream for DBCS data returned by the workstation environment does not contain any SOSI characters.

Using DBCS and mixed data fields

DBCS fields can only contain DBCS data. However, data can also be defined with a data type of MIX. The MIX data type allows both DBCS data and SBCS data.

For GUI clients, only the DBCS data type exists, not MIX. String data type supports SBCS and mixed DBCS and SBCS when using a DBCS code page.

Mixed fields in host environments (EBCDIC format) require special delimiter characters to identify a DBCS substring within the field. A single-byte, shift-out (SO) character in a mixed field indicates that the data that follows is DBCS. A single-byte, shift-in (SI) character in a mixed field indicates that the data that follows is SBCS.

Note: DBCS fields cannot contain SO and SI characters.

In most examples of mixed data, single-byte characters are shown surrounding double-byte characters. Each double-byte character is represented by 2 single-byte characters in the form of Dx, where x is replaced by a lowercase alphabetic character. The SO character is represented by the less than symbol (<) in the following examples of mixed data and has an actual hexadecimal value of 0E. The SI character is represented by the greater than symbol (>) in the following examples of mixed data and has an actual hexadecimal value of 0F.

Mixed data with surrounding SBCS data in EBCDIC format

```
ABC<DiDjDk>DEF
```

Mixed fields on the workstation (ASCII format) do not require control characters to delimit double-byte data. A double-byte character can immediately precede or follow a single-byte character. For example:

Mixed data with surrounding SBCS data in ASCII format

```
ABCDiDjDkEF
```

To enable external source files to be moved between the host (EBCDIC) and the workstation (ASCII), VisualAge Generator Developer removes or inserts the SOSI characters around DBCS and mixed data in the external source format file on import or export. Do not remove these SOSI characters: mixed literals and mixed data in table contents that contain adjacent groups of DBCS data (not separated by blanks) have all SOSI characters removed on import; however, only one pair of SOSI characters is inserted around the DBCS data on export. For example, on import

```
ABC<DiDjDk><DlDmDn>DEF
```

becomes

```
ABCDiDjDkDlDmDnDEF
```

On a later export,

```
ABCDiDjDkDlDmDnDEF
```

becomes

```
ABC<DiDjDkDlDmDn>DEF
```

The above literal has different lengths from the import and export operations. If the mixed literal is used as a source operand to be moved to a high-level VisualAge Generator structure, such as a record, the result might be unpredictable.

Note: The use of a mixed literal in this way is not generally recommended, and it is your responsibility to ensure that design of the program is portable between the two different environments where the program runs.

If a DBCS or mixed map constant field is used as a header for multiple columns, the alignment of the header might appear different with or without the SOSI characters. You must align the header for display in both environments or define the header as separate fields.

Using DBCS names

VisualAge Generator Developer supports DBCS names for any part with a name that can be longer than 18 single-byte characters. A valid DBCS name must meet the following conditions:

- The DBCS part name cannot be a mixed name, using both SBCS and DBCS characters.
- The DBCS part name cannot contain a DBCS blank.
- The maximum number of DBCS characters in a name is shown in Table 25:

Table 25. The Maximum Number of DBCS Characters for Each Part Name

Part Name	Maximum
Function	8
Record	8
Data item	15

- A DBCS name must contain at least 1 DBCS character that does not have a SBCS equivalent (non-42nd-ward DBCS character). The only valid SBCS-equivalent (42nd-ward) DBCS characters are as follows:
  - Double-byte A through Z (.A - .Z)
  - Double-byte 0 through 9 (.0 - .9)
  - Double-byte **@**, **#**, **$**, _ and **-** (hyphen)

Double-byte lowercase characters a-z are folded to double-byte uppercase A-Z when used in a DBCS name.

Note: A 42nd-ward DBCS character contains Hexadecimal 42 in the first byte when translated to EBCDIC.

Table 26 shows DBCS names that are valid and not valid:

Table 26. Valid DBCS Names vs. Invalid DBCS Names

Valid DBCS Names	Not Valid DBCS Names
.CDi.B	.A.B.C
DiDjDk	AB.C

Note: To avoid aliases being assigned during program generation and to improve the readability of the generated program, follow these standards:

- Do not use a DBCS name for a function or record.
- Do not use a DBCS name for a data item name if your program contains SQL functions.

Defining data

Through data definition you can define data items, records, and tables.

Defining records

A record is defined to VisualAge Generator with a record specification and a list of data items. You can define DBCS and MIX type data items in the Record Editor window. You can specify a data type of MIX to indicate that the field contains SBCS and DBCS data, and a data type of DBCS to specify that the field contains only DBCS data.

You do not have to account for the possibility of DBCS subfields when specifying the length of a MIX data item. The length specified is the number of single-byte characters that the field can contain. The number of bytes for a MIX type must equal the length you specified on the Record Definition window.

When defining a DBCS data item, specify the length in DBCS characters. The number of bytes for a pure DBCS field is double the length of DBCS characters in the field. If the length is changed, the number of bytes is automatically calculated.

A record is composed of data items that can be substructured. Both MIX and DBCS data types can be substructured. A data item of type MIX can be below

or above (have a lower or higher level number) than data items of type CHA or MIX. A data item of type DBCS can be substructured only under a CHA data type.

Note: Moving data into a CHA field that has a MIX or DBCS field substructured on or over the CHA field can result in incorrect data. Moves to MIX type fields are validated. Moves to CHA type fields are not validated unless the source is defined as MIX type. See "Data movement processing" on page 222 and Figure 26 on page 223, which depicts mixed data movement in structures.

Defining tables

You can specify table columns and table contents as DBCS.

Defining table columns

Specification of mixed and DBCS table columns is similar to record and data item definition. See "Defining records" on page 216 for a more detailed description of mixed data in substructures.

Defining table contents

The Table Contents function of the Table Editor enables you to enter DBCS data into table columns that have been defined as type DBCS and MIX data into table columns that have been defined as type MIX.

The fold table contents specification option is used to fold MIX type data columns from lowercase characters. Only single-byte portions of MIX data are folded; DBCS columns are not folded.

If you generate a table for use in host environments, the table contents have SOSI characters inserted around DBCS substrings. If a table column is not defined with a length that allows enough space for SOSI with specific contents entry, the system will truncate the contents entry.

Defining data items

The Data Item Editor enables you to specify a data type of DBCS or MIX for a data item. You can then enter a description of the data type in the **Description** field of the Data Item Editor window.

Defining prologs

The Prolog editor supports the entry of record, table, and program prologs. Mixed data is valid on the Prolog window.

Defining GUI clients

In the composition editor, you can type the DBCS name directly or display and select a DBCS name from a select list.

You can connect to DBCS and MIX data items

GUI text field data types

The following data types support DBCS when running in a DBCS environment:

String Supports single-byte and double-byte characters. String is used for either SBCS strings or mixed strings.

DBCS String
Supports double-byte characters.

Data item connections

The following considerations apply for DBCS data:

- String data is compatible with CHA and MIX data. DBCS String data is compatible with DBCS data.
- Any connection is allowed at definition time.
- At run time, if single-byte characters are typed in a DBCS String field, you will receive an error when the data is moved to the data item.
- At run time, if double-byte characters are typed in a field in a visual part, and that field is connected to a CHA data item, you will receive an error when the data is moved to the data item.

Defining maps

The Map Editor allows you to specify general options/preferences for your VisualAge Generator maps and allows you to specify the devices you will use with your program.

Selecting DBCS devices

If you plan to enable the program user to enter DBCS or mixed data on the map you are defining, you must select the DBCS display device (IBM 5550D) on the **Devices** page of the **Map Properties** notebook. Any program that displays a map defined for an IBM 5550D must be run by VisualAge Generator Server for MVS, VSE, and VM using one of the IBM DBCS devices.

If you are defining a printer map and plan to print DBCS or mixed data, you must select the DBCS printer device (IBM 5550P). Program output produced from a map defined for an IBM 5550P must be directed to a printer that supports DBCS printing.

A single map cannot support both a DBCS device and a non-DBCS device. One or the other must be specified. If a DBCS device is replaced with a non-DBCS device, any reference to field outlining is removed and all mixed and DBCS fields are changed to single-byte character set fields.

SOSI take position

SOSI take position enables you specify that SOSI characters take a position when printing mixed data from a program running on an MVS VM, VSE, or AS/400 system. SOSI take position is the default value for a new map.

If SOSI take position is selected, the generated map group program inserts a blank character before each shift-out character and after each shift-in character when print lines are formatted. If SOSI take position is not selected, blanks are not inserted. Select SOSI take position when you are directing output to a printer that removes SOSI characters from the print line. Deselect SOSI take position when directing output to a printer that prints blanks in place of SOSI characters.

Two buffer sizes are used for printing maps. If the DBCS invocation parameter is specified, or you are using an IBM DBCS device, then a buffer size of 650 bytes is used. This larger buffer is required to hold the additional data for blank insertion around SO and SI characters and the added attribute data for field outlining.

SOSI take position is ignored when testing or running programs on workstation systems.

Using the map editor

Maps can be defined for running in a DBCS system. Considerations for defining maps using DBCS and mixed character fields are described in this section.

Defining DBCS and mixed fields

All DBCS, mixed, and SBCS fields are defined in a single map editor window. The map editor window checks that data entered in these fields is in the correct format; for example:

- SBCS characters cannot be entered in a field defined for DBCS, and DBCS data cannot be entered in fields defined as SBCS.
- If you specify an initial value for a field using the Variable Field Properties window, it must be a valid character type for the defined field.
- DBCS fields are kept at an even length.
- Some workstation editor functions, such as clipboard operations and insert or delete, can incorrectly alter DBCS or mixed fields. The map editor validates each field if a map is saved or previewed.

It is not recommended that you edit DBCS maps on a non-DBCS device. If a non-DBCS device is used for editing, the validation described above is not performed because a DBCS code page is not in use.

DBCS map definition considerations

The following rules apply when defining a DBCS or mixed field:

- DBCS and mixed fields cannot wrap at the beginning of a map.
- DBCS and mixed fields cannot wrap if the width of the map is less than the width of the DBCS device selected.
- If the device selected for the map is a printer the following is true:
  - Mixed variable fields cannot wrap from one line to another
  - DBCS variable fields that wrap from one line to another must begin in an odd column
- If all data contained in a field is DBCS, you should use a DBCS field. If there is a possibility that mixed data might be contained in a field, the mixed field should be used. It is recommended that SBCS field be used when mixed or DBCS data is not present.

You can specify the map message field EZEMSG as a mixed field.

Defining variable field edits

The following rules apply when defining variable field edits:

- DBCS and mixed fields can be left-aligned, right-aligned, or not justified.
- All fields in a map array must be of the same type (DBCS, mixed, or SBCS).
- The only valid fill characters for DBCS fields are blanks and nulls. Any keyable single-byte character is valid as a fill character for mixed fields.
- The fold option is not applicable to DBCS fields. If folding is specified for mixed fields, only single-byte portions of the mixed data are folded.

Field attribute definition

Attributes pertaining to mixed and DBCS variable fields are discussed in this section.

Note: VisualAge Generator Developer supports color and extended attributes for all devices that provide these attributes.

Field outlining

To specify outlining attributes for specific fields, from the Map Editor window, select **Field Properties** from the **Define** pull-down, and then select the **Attributes** notebook tab. The controls used to define the outlining attributes are only visible for maps that are defined for IBM DBCS devices. With field outlining you can draw horizontal and vertical bars around a map field. The outlining attribute that you define is applied to the entire map field. For example, a line cannot be drawn around only the first 3 characters of a 5-character map field.

If you replace a DBCS with a non-DBCS device any reference to field outlining is removed.

The field outlining controls enable you to specify the following values. The default value is None (no outline).

Box Turns on all four outlining attributes

Over Draws a line over the field data from the attribute position that starts the field to the attribute position that ends the field

Under Draws a line under the field data from the attribute position that starts the field to the attribute position that ends the field

Left Draws a vertical line in the attribute position that starts the field

Right Draws a vertical line in the attribute position that ends the field

Note: Field Outlining can be specified with any combination of valid extended attributes. A field can be outlined, have a color, and use one of the supported extended highlighting attributes.

Maps generated for the OS/400 environment should contain blanks in row 1 columns 1 through 4, if one or more fields on the map have field outlining attributes. Row 1 column 4 can contain a field attribute.

Defining programs

This section covers the specific issues you must consider when you define programs that contain DBCS.

Defining statements

You can specify mixed and DBCS literals when you define statements. You can specify a mixed literal by enclosing the literal data in quotation marks and having both SBCS and DBCS characters within the data. If DBCS characters are not present, the literal is assumed to be a character literal. If single quotation marks are used, the SBCS data in the literal is folded. DBCS characters or data enclosed by double quotation marks is not folded.

You can specify a DBCS literal by enclosing DBCS characters in quotation marks and prefacing the first quote with a G. SBCS characters, including SBCS blanks, are not permitted between the G and the second quotation mark. The preceding G forces the literal to be a DBCS rather than a mixed literal; therefore, a DBCS literal must be in the sequence: G'DiDjDk'.

Statements in functions or program flow can use DBCS names, DBCS and mixed literals, and comments.

SOSI characters in literals are inserted when a program is generated for an MVS VM, or VSE environment or when a part is exported.

Data movement processing

Data item movement in a VisualAge Generator program occurs as a result of the ASSIGNMENT, MOVE, MOVEA, and RETR statements.

DBCS Data Items

DBCS data items can only be moved from or to another DBCS data item. When a DBCS data item is moved to a DBCS data item of a different length, padding or truncation occurs. If the target item length is longer than the source data item length, the target is padded with blanks. If the target data item length is shorter that the source item length, the source data is truncated.

Mixed Data Items

Mixed data items can only be moved to or from another mixed data item or a character data item.

When you move from or to a mixed data item, mixed data validation is performed. If the target string results in an incorrect mixed string, VisualAge Generator ends processing.

When a mixed data item is moved to a mixed data item of a different length, padding or truncation occurs.

If the target item length is longer than the source data item length, the target is padded on the right with single-byte blank (X'40') characters. If the target item length is shorter than the source item length, the source data is truncated. In System/370 and OS/400 environments, if the place of truncation is in a DBCS substring, only enough data is moved to result in a complete DBCS substring that contains at least one DBCS character. Unoccupied positions in the target that result from DBCS substring truncation are filled with single-byte blank (X'40') characters.

When testing or running a program on a workstation system, mixed values do not contain SOSI characters. If a truncation point is within a double-byte character when testing or running on a workstation system, that character is truncated and the moved value is padded with a single-byte blank (X'40') characters.

Figure 25 on page 223 illustrates truncation and padding in host environments resulting from mixed data movement:

```
Data in a source mixed field of length 14 = ABC<DiDjDk>DEF

Target mixed field of length
     3 = ABC
     4 = ABCb
     5 = ABCbb
     6 = ABCbbb
     7 = ABC<Di>
     8 = ABC<Di>b
     9 = ABC<DiDj>
    10 = ABC<DiDj>b
    11 = ABC<DiDjDk>
    12 = ABC<DiDjDk>D
    13 = ABC<DiDjDk>DE
    14 = ABC<DiDjDk>DEF
    15 = ABC<DiDjDk>DEFb
    16 = ABC<DiDjDk>DEFbb


where b indicates a single byte blank (X'40') character.
```

Figure 25. Mixed Data Movement - Truncation and Padding

Only the strings used directly in the processing are checked for a valid mixed string; therefore, a mixed string that is not valid might be created by the MOVE statement for a different data item. The mixed string that is not valid might not be detected while the MOVE statement is processing. This is especially possible when data items are defined in substructures,

Figure 26 shows mixed data movement in substructures in MVS VM, and VSE environments. MIXV1 is a level 10 data item that is substructured into two level-20 data items, MIXV2 and MIXV3.

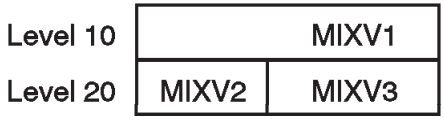

Figure 26. Mixed Data Movement in Substructures in MVS and VSE Environments

In Figure 26 the following is true:

- MIXV1 is a mixed item of length 20. The data contained in MIXV1 is: <DiDjDkDlDmDnDpDq>.
- MIXV2 is a mixed item of length 4 substructured under MIXV1.
- MIXV3 is a mixed item of length 16 substructured under MIXV1.
- Using the VisualAge Generator MOVE statement, MOVE <Dz> TO MIXV2.

- The result is <Dz>DjDkDlDmDnDpDq>, for MIXV2. Therefore, the move is considered valid. Note that the data in MIXV1 is not valid but is not detected.
- The result of a move from MIXV1 to another string of length 20 is not correct because MIXV1 currently contains <Dz>DjDkDlDmDnDpDq>.

Data item comparison processing

The following data item comparisons in a VisualAge Generator program occur as a result of the IF/WHILE, FIND, and RETR statements:

Mixed data items

A mixed data item can only be compared to another mixed data item or to a character data item. For comparisons, a left-to-right byte-to-byte logical comparison is done.

If the data item lengths are different, the shorter item is padded with single-byte blank characters (X'40') to the length of the longer item prior to the compare. Then, the two strings are compared for the length of the longer data item.

DBCS data items

A DBCS data item can only be compared to another DBCS data item. For comparisons, a left-to-right byte-to-byte logical comparison is done.

If the data item lengths are different, the shorter item is padded with double-byte blank characters to the length of the longer item prior to the compare. Then, the two strings are compared for the length of the longer data item.

Using mixed literals as CALL statement arguments

Avoid the use of mixed literals as CALL statement arguments because the literal has a different length when it is generated for ASCII-based systems than when it is generated for EBCDIC-based systems. The SOSI characters are present in the EBCDIC literal, but not in the ASCII literal.

To avoid this problem, define a mixed variable with the same length as the corresponding called parameter definition in the called program. Specify the mixed variable as the CALL argument and move the literal value to the variable prior to the CALL.

Testing programs

The test facility supports DBCS and mixed data on your maps for the IBM DBCS devices. All fields with the data type MIX are validated before being sent to the device. In addition, all trace statements displayed to an IBM DBCS device that refer to data items of data type MIX are validated before being

displayed. A mixed data item that spans lines because of the length of the data item is split and validated according to mixed data splitting rules.

VisualAge Generator Developer test facility trace assumes that data being retrieved from an SQL row with a data type of CHA might contain mixed data; therefore, for the trace only, the data item is assumed to have a data type of MIX. It is recommended that data containing mixed notation defined to an SQL row as Char be moved immediately to a VisualAge Generator Developer data item with type MIX. This assures mixed validation during runtime.

Understanding relational database support

For general information on SQL and relational databases, see "Chapter 2. Developing SQL programs" on page 19.

DB2, DB2/VSE, and SQL/DS VM support mixed data in table and column names. If a DBCS national language version of DB2/2, is installed on the workstation, DBCS and mixed data are supported in DB2/2

Character constants can be used in SQL statements. Use G notation (for example, G'DiDj') for DBCS constant fields. Use character literals (for example, 'eeeDiDjee') for mixed strings. VisualAge Generator inserts SOSI characters around DBCS strings in literals when the literals are generated for MVS, OS/400, VSE, and VM environments.

VisualAge Generator allows mixed data and SOSI notation for DBCS constants to be entered in the table and column name fields and in SQL statements.

DB2, DB2/VSE, and SQL/DS VM do not support a mixed data type. Instead, they support a system DBCS invocation parameter that allows any character column to contain mixed data.

VisualAge Generator does not allow data items in SQL rows to be defined as mixed because DB2, DB2/VSE, and SQL/DS VM do not support the mixed type. However, in VisualAge Generator programs, data can be moved between character columns in SQL rows and mixed fields in a map or working storage. It is recommended that you move any mixed data retrieved from the database to a data item defined as mixed in VisualAge Generator.

Specifying SQL row records

When an SQL row is being defined in record definition on an IBM DBCS device the table name can be specified using mixed data.

Defining SQL row data items

Data item definition is different for an SQL row record than for the other record organizations. The items represent columns in a relational table instead of a record structure. In addition to the data item name, specify the column name for the item as it is known to the database manager. You cannot specify level and occurs values for items in an SQL row record.

Relational databases support the use of mixed data for their column names. VisualAge Generator enables DBCS device users to enter mixed SQL column names on the SQL Row Item Usage Data window of Record Definition facility. VisualAge Generator does not support mixed data item names.

The type field for a mixed SQL data item should be specified as Char. VisualAge Generator only supports valid SQL types for its data type. Any mixed data contained in an SQL Char field received from the relational database is assumed to be valid mixed data. Mixed data being moved from VisualAge Generator to a relational database should be moved from a data item or map field having the data type Mixed to an SQL row data item having type Char. Movement of data between Char and Mixed data items is validated using the mixed validation rules. See "Data movement processing" on page 222 for details on mixed data movement.

Using mixed SQL statements

You can define SQL statements using mixed notation. The SQL Statement Editor window of the Record Editor enables the use of mixed notation in creating the selection conditions. All fields that can be modified support the entry of mixed data.

If the program is running on a MVS, VM, VSE, or OS/400 system, SOSI characters are inserted as required into the SQL statement text.

Preparing programs with DBCS support

When you compile a program, map group, or table that uses DBCS or mixed data, you must include the DBCS compiler option.

If you are using CICS, you must be on CICS/ESA Version 3 Release 1 Modification 1 or later. You must include the DBCS option on both the CICS translator and the COBOL compile steps.

OS/400 DBCS customers using RISC hardware can use OS/400 Version 3.6 or later. OS/400 DBCS customers using non-RISC hardware must use OS/400 Version 3.2 or later.

Appendix A. Program design techniques

This appendix describes easier ways to use particular functions of VisualAge Generator during the program coding phase. The techniques shown here have significant benefits for performance, storage, or ease-of-use.

Invoked functions

Invoked functions should be defined for any group of identical statements that appear in multiple functions. This has four advantages:

- Conserves real storage at run time
- Saves keying repetitive lines of statement definitions
- Conserves Repository/ENVY library storage during definition and testing
- Simplifies maintenance

Data items referenced within an EXECUTE function must be qualified if they appear in more than one structure (map, record, or table). You should consider qualifying all data item references if it is possible that the data item might be defined in another structure in the future.

Recursive functions

A function is recursive if it invokes itself or another function it is nested under (for example, A invokes B, and B invokes A).

In most cases it is better to use a WHILE statement rather than a recursive function to implement processing loops.

Notes:

1. Do not use function invocation statements for unconditional flow, transfer, or return processing. Use the special function word EZERTN for an immediate return to the invoking function. Use the special function word EZEFLO and unconditional branch flow statements for “go to” or transfer processing.
2. Recursive program calls (for example, A calls B, and B calls A) are not supported by VisualAge Generator. Recursive use of functions is supported.

Map edit functions

Edit functions can be useful for performing complex edits on map fields. Figure 27 on page 228 shows the general format of an edit function:

```
SINQE01:
  /*****************************************************************
  /* Perform edits here.  If o.k. then EZERTN
  /* Else set up for error.
  /*****************************************************************

  /*****************************************************************
  /* Following is sample code to set up for error.
  /*****************************************************************
  SET MAP3.FIELD1 MODIFIED BRIGHT; /* Set FIELD MODIFIED so it is
                                   /* edited again and BRIGHT to
                                   /* indicate it is in error
  IF EZEMNO NE 0;                  /* Figure out if this is
                                   /* the first error
     EZERTN;                       /* Prev. error detected, so exit
  ELSE;                            /* This is the first error
     MOVE 34 TO EZEMNO;            /* Set up correct error message
     SET MAP3.FIELD1 CURSOR;       /* Put cursor under this field
  END;
  EZERTN;
```

Figure 27. General format of an edit function

This function does the following:

- Returns control if edit errors are not detected.
- If edit errors are detected the following occurs:
  - The field is highlighted
  - The map field is set MODIFIED to ensure that it is edited again.
  - The EZEMNO special function word is checked to determine if this is the first error to be detected. If it is, the proper error message is set up and the cursor is placed under this field.

EXECUTE functions

EXECUTE functions are typically used in the following situations:

- Initialization code
- File error processing and message setup
- Separating unique processing statements from generalized I/O processes
- Loop control
- A place holder for a central FLOW control point
- Function invocation

The choice of whether to include logic statements as part of the statements preceding an I/O function or to move them to an EXECUTE function that is run immediately before the I/O function is a matter of programming style and makes little difference to program efficiency.

MOVE statement

The MOVE statement has two different forms. One moves data from one data item to another data item. The other form moves data between two structures (that is, records or maps) by moving all the data items that have matching names in the structures.

Data moved between two structures with a single statement is called a structured MOVE or a move corresponding. These structures can be records or maps. Level-77 items are not included in a record structure and are not moved. When you generate your program the structured move is expanded into the multiple MOVE statements. The multiple MOVE statements are equivalent to the structured move.

The expansion is done internally and is not printed in your generation listing. If you are using nonshared data items, then items that have the same name in different structures can have different characteristics. If they are compatible, data conversion is the same as for a single item move. If they are not compatible, an error message is issued during testing or generation.

Moving data items between maps and records

By naming a map and a record on your MOVE statement, you can easily move the matching data items between the two. When moving from a record to a map, you should be sure the record data can be displayed. If a character data item in a record contains data that cannot be displayed, it can cause terminal errors to occur when moved to a map. If a field having the same name exists in both the record and the map, and it is defined as binary or packed in the record, then it must be defined as numeric on the map.

Moving between records

When moving entire records, it is better to use a MOVE statement between the two high-level data items of the records rather than doing a move corresponding. Both accomplish the same thing, but the data item MOVE statement results in just one move instead of a move for each data item. If a high-level data item is used, be sure that the data items defined in both structures match in length and type because data conversion is not done.

Similarly, if you are moving part of your record to another record, it is more efficient to move the highest level structures possible in the records.

Figure 28 on page 230 shows an example of moving between records. If the move corresponding is done, nine moves are performed instead of one if the high level data items are used.

```
     DO NOT                    DO

MOVE REC1 TO REC2      MOVE REC1.ALL1 TO REC2.ALL1


    REC1                              REC2

    ALL1   LEN  256                   ALL1   LEN  256
      STR1   LEN  100                   STR1   LEN  100
        FIELD1 LEN  030                   FIELD1 LEN  030
        FIELD2 LEN  030                   FIELD2 LEN  030
        FIELD3 LEN  040                   FIELD3 LEN  040
      STR2   LEN  156                   STR2   LEN  156
        FIELD4 LEN  050                   FIELD4 LEN  050
        FIELD5 LEN  050                   FIELD5 LEN  050
        FIELD6 LEN  056                   FIELD6 LEN  056
```

Figure 28. An example of moving between records

If you are rearranging the order of the data in the records at the same time you are moving the data, then using the move corresponding is easy to define because it is a single statement. It can be more efficient (but less convenient) to define the individual MOVE statements. Structures with identically named items at different levels result in some redundant moves if you use the move corresponding. It is not determined whether the data has already moved through a higher level item.

Table Data

Direct addressing of table data items

A specific item in a table can be referenced in a processing statement by using subscripts or numeric literals. For example, TABLNM.COLNAM(43) addresses the data value in the 43rd row of the column name COLNAM in the TABLNM table. This method of obtaining data from a table is more efficient than using either the FIND or RETRIEVE statements. This technique can be especially useful if you are substituting a character string for a numeric code (such as state names for state numeric codes).

Table searches with the FIND or RETRIEVE statement

Always order the data in the column that you are searching by frequency of use. VisualAge Generator searches tables by examining each entry beginning with the first entry until a match or the end of the table is encountered.

Both the FIND and RETRIEVE statements set the EZETST special function word to the row number of the matching data value. One technique is to define the FIND statement in the FLOW of a function. Defining a FIND statement in FLOW enables you to specify a function to run if the item is

found. Another function can be specified as the FALSE condition. This eliminates the overhead of coding an IF statement to test the EZETST special function word to determine if the item was found.

A table searched with a FIND statement can be used in the flow section of a program instead of an IF statement that checks multiple 'OR EQUAL' conditions. Refer to Figure 29. The common definitions appear in **bold**.

```
        PUPD100:   --- execute ----
     (flow)                          (flow)
                        or
IF WS.DAY EQ 'MON'                FIND WS.DAY TAB.DAY PUPD200 PUPD300
 OR WS.DAY EQ 'TUE'
 OR WS.DAY EQ 'WED'
 OR WS.DAY EQ 'THU'
 OR WS.DAY EQ 'FRI';
   PUPD200;
ELSE
  PUPD300;
END;

        PUPD200:     PROCESS WEEK DAY DEPOSITS
                      .
                      .
        PUPD300:     PROCESS WEEK END DEPOSITS
                      .
                      .
```

Figure 29. Using FIND to do an IF

Defining tables

In the CICS and IMS environments, tables can be defined as shared or non-shared tables. If a shared table is loaded, all the transactions running in the same region use the same copy of the table.

For tables that are not modified by programs, specify the shared option to reduce memory usage and table load time. For tables that are modified by programs, you can also use the shared option to use the table as a shared communications area between multiple programs. Note that shared tables can only be modified in CICS environments. For more information, see "Communicating between multiple CICS transactions" on page 185.

The **Keep After Use** language element controls when a table is loaded into storage. **Keep After Use** and **Resident** elements together control when a table is deleted from storage.

Use the Table and Additional Records List in the Program Editor to turn off **Keep After Use** for tables that are conditionally used within a program. An example is a map edit table used on a map shown from a CONVERSE function only when the user requests a rarely used function.

Using the Table Editor, specify **Resident** for shared tables that are used frequently in high performance, high volume CICS transactions. The table remains loaded until CICS comes down or a new copy of the table is requested using the VisualAge Generator Server for MVS, VSE, and VM or VisualAge Generator Server new copy function in the CICS utility.

Changing table contents while running the program

A program can modify table contents at run time using processing statements such as MOVE. Such changes are temporary. They are lost when the table is deleted.

If you want to provide a private copy of the table for each program to use that can be modified and not deleted until the program ends, you must do the following:

1. Specify **Keep After Use**, the default, using the Table and Additional Records List function of the Program Editor.
2. Using Table Definition, remove the **Shared** attribute to define the table as single-user.
3. Do not run the program in segmented mode.

On CICS systems, you can use a table as a shared communications area between all active users of a program by doing the following:

1. Using Table Definition, define the table as **Shared**.
2. For CICS for MVS/ESA or CICS for VSE/ESA environments, mark the table load module as RESIDENT in the PPT.

When marked resident, the shared table is not deleted, even when the program is running in segmented mode. For more information about using a table as a shared communications area, see "Communicating between multiple CICS transactions" on page 185.

For IMS/VS and segmented programs, modification of shared tables is not supported. Because IMS/VS programs run in segmented mode, modifications of single-user tables are lost at each CONVERSE function. Resident is not supported for segmented programs.

Message tables on OS/400

On OS/400 execution systems, message tables are implemented as native OS/400 message file objects (*MSGF). This has the advantage of providing second-level message text when viewing the messages from a display map and sharing the message file with other programs even if they are of another

high-level language form. However, the disadvantage is that message table rows and columns cannot be referenced by program logic in VisualAge Generator because they are not actually tables as other table types are managed at runtime. This is a program execution consideration only, for OS/400, user messages are still defined within message table parts in the Repository/ENVY library.

Initializing data fields

Data items are initialized as follows when a program starts:

- Working storage items, including level-77 items in the primary working storage record, are set to blank if they are CHA, MIX, or DBCS; 0 if they are NUM, NUMC, PACK, PACF; and binary 0 if they are BIN or HEX
- Implicit variables are treated the same as level-77 working storage items. These are not created in test facility until they are encountered while testing.
- All map data items are set to blank if they are CHA or DBCS and to 0 if they are NUM.
- If DXFR or XFER statements were used to transfer to your program and working storage was passed, then the first part of your program working storage contains the passed working storage. If your working storage is shorter, the passed working storage is truncated. If your working storage is longer, it is padded with blanks (test facility only). For generated COBOL programs, the working storage record is initialized based on the type of data before the passed working storage data is moved into the record. Therefore, if your working storage is longer than the passed working storage, the extra area is initialized based on type.

  For generated C++ programs, unpredictable results will occur if the passed working storage record is shorter than your program's working storage record.
- Data items in serial, indexed, relative, DL/I segment, and SQL row records are not initialized. You must define a SET record EMPTY or specify the /INITRECD generation option to initialize these records.
- Initialization of data items in working storage records, other than the primary working storage record, is controlled by the /INITADDWS generation option.

Note: Record initialization is done at the lowest (deepest) level of nesting. Therefore, if you have items of different data types nested under each other, you can end up with blanks in NUM, NUMC, PACK, PACF, or BIN data items. You should ensure that you set them to numeric values before using them in calculations.

If your program performs a 'SET mapname EMPTY', the data items in the map are blanked or zeroed according to type.

The following are suggestions for initializing individual data items:

- Use MOVE ' ' TO REC.FIELD to blank out a CHA field. It is only necessary to put one blank between the quotation marks. The length of the item does not matter.
- Use the following statement:

  ```
  MOVE 0 TO REC.FIELD
  ```

  to initialize NUM, NUMC, PACK, PACF, or BIN data items to 0.
- Use the MOVE statement to put data values in the highest level data item possible in the record structure that initializes the data fields correctly.

Data types

The following are guidelines for defining data types:

- *Binary* data type is the most efficient for array subscripting and relative record IDs.
- When using binary data, try to use short *binary* positive numbers with no decimal places. 'Short' includes numbers with values less than 32,768 (or defined as four numeric digits), that can be resolved into a length of two bytes.
- If a numeric item is going to be used on a map as well as in a few calculations, define it as *numeric* instead of binary.
- *Numeric* data without decimal places is more efficient in calculations, moves, and comparisons than numeric data with decimals. If decimal places are required, try to be consistent in the number of decimal places across all items in a calculation. Addition or subtraction of binary numbers when all items in the statement do not have the same number of decimal places is inefficient. This addition and subtraction is inefficient because the decimals for the numbers are aligned to match the result before doing the addition or subtraction. It is more time-consuming to align decimals for a binary number than to align decimals for a numeric number.
- VisualAge Generator attempts to convert any data as it is moved to a data item defined with different characteristics. A severe error occurs only if this conversion is not successful. Decimal alignment is handled automatically. Data truncation warnings are generated during program test when binary fields are moved into smaller fields, numeric data loses significant digits on moves or calculations, or character data cannot fit into the target field.
- VisualAge Generator handles *numeric or binary* data with up to 18 digits, including decimal places.

Controlling flags and counters

Flags used by your program should be defined as CHA data items. They should be set or reset by using a MOVE, and tested using IF or WHILE statements. Implicit variables can be used for character flags. Numeric flags are less efficient to test because signs and decimal places must be considered prior to the compare operation.

Counters should be defined as BIN data items (no decimal places). Counters can be defined in working storage as level-77 items. NUMC performs better than NUM.

Addition or subtraction of two variables (or a variable and a literal), that are numeric and have the same number of decimal places, is performed without data conversion.

Subscripting data items

Subscripting is used to refer to a single occurrence of a data item defined with an occurrence (OCCURS) greater than one, or to reference a data item in a specific row of a table, or to reference a specific PCB using EZEDLPCB. If the data item has an OCCURS greater than one and no subscript is supplied, the first occurrence of the data item is used.

The following example shows subscripting a data item with an OCCURS greater than one. In the example, STATES is a record containing a data item called NAMES. NAMES has an OCCURS value of 3. STATES contains another data item called NUMB, and this data item contains a number.

To move the contents of the third occurrence of NAMES in the record, STATES, NUMB is first set to equal 3. The following statement moves the contents of the third occurrence of NAMES into a map item called MAP.STNAM:

```
MOVE 3 TO NUMB;
MOVE STATES.NAMES[NUMB] TO MAP.STNAM;
or
MOVE STATES.NAMES[3] TO MAP.STNAM;
```

The NUMB data item is the subscript that specifies which occurrence of NAMES should be used.

Subscripting a table item

Items in tables can be subscripted. For example, a table called STATES contains a single column called NAMES. There are 50 rows in the table and each contains the name of a state. A number from 1 to 50 can be stored in a data item called NUMB, and NUMB can be used to reference a row in the table.

If NUMB equals 5, the following statement moves the contents of the fifth row in the NAMES column to a map item called MAPA.STNAM:

```
MOVE STATES.NAMES[NUMB] TO MAPA.STNAM;
```

By changing the value of NUMB, you can reference any row in the STATES table.

Subscripting EZEDLPCB

In the example, the IMS PSB being used by the program has 5 PCBs. If you want to access the contents of the I/O PCB, use the following MOVE command:

```
MOVE EZEDLPCB[0] to IOPCB;
```

Record processing techniques

Duplicate Keys

Records with duplicate keys can exist in indexed files. Duplicate keyed records can be read or written only in the SCAN, SCANBACK, and ADD functions. UPDATE and DELETE functions only act upon the first record in a set of records with duplicate keys.

Records with duplicate keys or without keys can exist in DL/I databases. You can UPDATE or DELETE records retrieved with the SCAN function if you modify the DL/I call for the function to do SCAN FOR UPDATE.

Record locks

A REPLACE or DELETE function must be preceded by an UPDATE or SCAN FOR UPDATE function in the same program. The number of locks that can be held at the same time varies with the system.

If your system uses VSAM indexed files, the program can hold one lock per file. If a second read operation, within the same program with the UPDATE function, is run against the same file, the first record lock is released.

If your program uses DL/I databases, the program can hold one lock per database structure (PCB entry in the program PSB). The lock is released on the next I/O function for the same structure.

Locking rows in relational databases is discussed in “SQL locking” on page 55.

Generic keys

Generic key search support for indexed files can be satisfied by moving a partial key into the record key field and doing SET record SCAN against the record. The subsequent SCAN receives the next sequential record with a key greater than the partial key.

For example, if a file has 10 character keys with people's names, to start scanning for names starting with 'JON,' you should move 'JON' to the record key and do a 'SET record SCAN' to start scanning.

Generic key search support for DL/I databases is described in "Searching on partial keys" on page 98. Generic key search support for relational databases is described in "Chapter 2. Developing SQL programs" on page 19.

Controlling file I/O

The task of sending or receiving data from a file can require as little as one VisualAge Generator statement or it can require several functions to set up and perform. Some of the important file I/O considerations are discussed in this section.

Data set position

Data set position for relative or indexed files determines what logical record is presented to your program if it does a record SCAN. If the program has a position, a record SCAN retrieves the next sequential record from the data set and sets the data set position to the record just accessed. Each successful I/O for a record sets the data set position to the logical record obtained on that I/O. If an I/O is unsuccessful (NRF, EOF, LOK, and so on), your position in the data set is unpredictable.

At the start of a program, the program is positioned to the first record in each of your data sets.

The position of the file is reset to the beginning of the data set when you call a program that does I/O to the same data set, but uses a different file name.

For CICS recoverable files, the position of the file is reset to the beginning of the data set when one of the following occurs:

- A commit point
- A rollback
- A segmented CONVERSE.

If you want to maintain your position for scanning across one of these points, you must save a unique key and re-establish your position by doing an INQUIRY (or SET record SCAN for indexed files) to the record using that key.

If you have multiple records defined in the same data set by using the same file name on each record definition, then you should be aware that there is only one position per file name. For example, if you have three records (REC1, REC2, REC3) and you specified the same file name (ARFILE) on all three definitions and the data set you were testing with was V90.UCAT1.ARDSINDX, then doing an INQUIRY to any one (for example,

REC2) of the three records sets the position in V90.UCAT1.ARDSINDX for all three records. The data content of REC1 and REC3 does not change. However if you then scan REC3, you get the next record in V90.UCAT1.ARDSINDX after the last record that had position set on (the INQUIRY of REC2).

To have more than one position maintained in the same input data set in the same run unit, associate multiple records with different file names with the data set. In the data set, a separate file position is maintained for each logical file name.

Abnormal termination

When VisualAge Generator detects an error at runtime, it ends with a return code and error notification. The error notification identifies either the statement or function where the error occurred and the reason for the error.

For more information on abnormal terminations, refer to the *VisualAge Generator Server Guide for MVS, VSE, and VM* document.

File error handling

File I/O return codes are classified as hard or soft errors. The hard errors cause the program to end with error messages. The soft errors enable the program to continue if the user provides an error handling function for the I/O.

You should always define an error routine on your file I/O functions. A program abnormal termination can occur if you do not.

The choices you have for an error routine are a function, EZERTN, EZEFLO, or EZECLOS. While one of the special function words might work adequately during initial testing, you should normally include logic in your program to determine what type of file error occurred and send an appropriate message to the user. This means using a function to decode the error and set up the error message. An I/O error routine is invoked as an inline subroutine when an I/O error occurs. When the function completes, processing continues with the statement immediately following the I/O option. An I/O error routine works as an error exit. When the error routine completes, processing continues as directed by the flow stage of the process.

A program can continue after a file I/O error or a hard error if it has access to the error information. This enables programs to be written that can provide graceful degradation of service when some of their resources are not available. For example, if a file is full, a program could continue providing information from that file, or if there was a hard error on one file, the program could continue providing services that did not require that file.

The program can set a special function word, EZEFEC, to control continuation after hard errors. If this form of error handling is chosen, the program can test for hard errors and access error information. DL/I and SQL error information is returned in separate special function words. For other I/O, the return code is in the EZERT8 special function word. Refer to the *Programmer's Reference* document system for information on the format of EZERT8.

When file I/O is done for a program and the user has defined an error handler on the I/O function, the content of the EZEFEC special function word is checked to see if the program has requested special error handling. If the value of EZEFEC is equal to 0 (the initial value) then the program ends on hard errors. If the value of EZEFEC is equal to 1, the program continues on hard or soft errors.

Both mnemonic error codes and the value in EZERT8 can be tested after file I/O. The EZERT8 value is subject to system dependencies and should be carefully used.

Mnemonic error codes

The mnemonic I/O error values vary by I/O option and file type. Refer to the online help system or the *Programmer's Reference* document for the actual mnemonic error codes that are set for specific I/O options.

The UNQ error code is a subset of DUP and applies only to indexes that have been defined as unique. That is, if UNQ is set then DUP and ERR are also set. Therefore you should test the most specific error codes first. If UNQ is set, the operation was not successful (ADD or REPLACE of a record containing an index field key that is a duplicate of an already existing key in the file). If UNQ is not set, the operation was successful. DUP indicates that records with duplicate keys exist on the file. DUP can be returned on an input as well as an output option.

Example of an I/O error routine

An example of using an I/O error routine is shown in the following figure. This example assumes that you are not using a user message table. Assume that a key was obtained in the CONVERSE function for MAP2 and that an UPDATE function for that record is immediately performed in function PINQ080, which received some sort of file error. Therefore, the program branches to function PINQ180. The program attempts to determine the error condition, calls a function to correct the problem, and moves an appropriate message into the EZEMSG special function word field of MAP2.

This example assumes there is one index defined with duplicate keys. If it was not, the DUP code does not need to be tested because this program does

not ADD, and DUP on an UPDATE only indicates there are other records with this same key.

```
PINQ070: CONVERSE MAP2

PINQ080: UPDATE  CUSTREC       PINQ180
         MOVE CUSTREC TO MAP3;
         MAP3.TOTAL = MAP3.TOTAL + CUSTREC.ARBAL;
         ....
         ....
  FLOW-  PINQ090;

PINQ180: ---- execute ---
         MOVE 'CUST FILE ERROR, CONTACT HELP DESK' TO MAP2.EZEMSG;
         TEST CUSTREC NRF  SINQ190; /* NOT ON FILE
         TEST CUSTREC DUP  SINQ200; /* DUPLICATE RECORD ON FILE
         TEST CUSTREC LOK  SINQ210; /* SOMEONE ELSE HAS LOCK
         TEST CUSTREC DED  SINQ220; /* POTENTIAL DEADLOCK DETECTED

  FLOW-  PINQ070; /* REDISPLAY THE MAP

SINQ190: MOVE 'NO SUCH CUSTOMER EXISTS' TO MAP2.EZEMSG;
         EZERTN;

SINQ200: MOVE 'ANOTHER CUSTOMER HAS THE SAME CODE' TO MAP2.EZEMSG;
         EZERTN;

SINQ210: MOVE 'CUSTOMER RECORD IS IN USE, TRY LATER' TO MAP2.EZEMSG;
         EZERTN;

SINQ220: CALL EZEROLLB; /* RELEASE LOCKS TO BREAK DEADLOCK
         MOVE 'CUSTOMER RECORD IS IN USE, TRY LATER' TO MAP2.EZEMSG;
         EZERTN;
```

Figure 30. Example of an I/O Error Routine

The example shows the following:

- If a potential deadlock condition is detected, the program calls EZEROLLB to release all locks so that the other program that is waiting can proceed. The program user should be notified of this.
- If none of the TEST statements are true, the program should issue a message to the program user that indicates they should call their help desk with information about what they were doing. There is limited problem determination data that the program can access other than the program and function names.
- The error messages are set up by moving literals to EZEMSG. Because these messages are exception messages, they might be considered for a user message table. The messages are then set up by moving the correct message number to EZEMNO.

Defining partial and floating maps

Partial maps are smaller than the standard size of the screen for a particular device. Partial maps can be defined and positioned so that more than one map can be shown on a screen at the same time. A floating map has no fixed starting location. In other words, floating maps can appear in more than one location on a screen or printer. The following discussion of partial and floating maps applies primarily to maps that are defined to display on your screen. For information on map definition functions for printer maps see "Defining printer maps" on page 242.

Floating maps

A floating map does not have a fixed starting location. Floating maps are identified by the **Floating map** check box on the **Layout** page of the **Map Properties** notebook. When you select **Floating map**, the starting line is set to **Next** and the starting column is set to **Same**.

A floating area is defined by the map group and device, using the Map Group Editor. Floating maps appear without any consideration to any partial fixed maps or full fixed maps that might be on the screen. The only requirement for floating maps is that they fit the floating area defined for the device in the program's map group.

Floating maps occupy the next available space in a defined floating area not already occupied by a floating map. If there is no room at the bottom of the floating area for a new map, all floating maps are deleted (erased), and the floating area is considered empty. The new map is positioned on the first line of the floating area.

If a floating map overlays a fixed map, only that part of the fixed map that is in the floating area is overwritten by the floating map. The rest of the fixed map remains intact. Unpredictable results can occur when fixed maps are overlaid by another variable field.

Only one map can be placed on a given line (that is, side-by-side maps are not supported). If a floating area is not defined for a device, the floating area defaults to the full device size.

For IMS/VS, a floating map is only valid for printer maps. A floating map cannot be used for display devices in IMS/VS.

Defining a presentation area for floating maps

You can define a presentation area for floating maps defined within a map group from the Map Group Editor window. This list contains all devices supported by maps within the map group and shows the current floating area definition (if any) for each device.

Define the floating area size by setting the lines and columns to nonblank, nonzero values within the ranges shown. For the position of the floating area, set the starting line and starting column to nonblank, nonzero values within the ranges shown. All four settings must either be blank or all must contain numbers within the ranges shown. If the values are all blank, then a specific floating area has not been defined for this device and the entire device is considered as the floating area.

Within a floating area, floating maps are displayed one after another, from the top of the floating area to the bottom. Floating maps cannot be displayed side by side. Usually, the floating area is defined with the same depth for all display devices for a map group. Different floating areas for display devices supporting the same set of maps within the group are of little use. It is possible to use different floating areas if floating maps are conversed during program execution. If the floating area is full, you must converse one of the maps contained in the floating area.

For IMS/VS, a floating area is only valid for printer maps. Do not use a floating area for display devices in IMS/VS.

Partial maps

Partial maps are maps that do not have a depth equal to the size of the physical screen. More than one of these maps can be shown on the screen at one time if their position and depth do not overlap. Side-by-side maps are not supported.

Fixed maps have a specific starting line number. A fixed map overlay occurs if a partial map appears and exactly overlays an existing fixed map on the screen (same starting line and depth). When a fixed map overlay occurs, VisualAge Generator replaces the existing map without erasing the panel. If you do not want this to happen, code a SET map PAGE statement before the second (partial) map is displayed or is conversed, which causes the panel to be cleared before the partial map appears.

Fixed maps that do not overlap any other fixed map can appear on the same display in any order.

For IMS/VS, the screen is cleared before each CONVERSE so that only a single partial map can appear.

Defining printer maps

Many of the map definition functions associated with display device maps are applicable to maps that are defined for printers. However, because these output devices support different functions, some differences exist.

When VisualAge Generator processes a DISPLAY option for a printer map, the output is sent to the resource associated with the internal file name EZEPRINT. You specify this association when starting the test facility or when generating the program. You can also dynamically change the printer name at run time by moving the name of the printer into EZEDESTP before the DISPLAY I/O option.

Fixed position printer maps

Fixed position printer maps can start on any specified line number, provided the total depth and offset do not exceed 255 lines.

Unlike display maps, the order in which printer maps are printed affects the output. VisualAge Generator monitors the current line number for the print output. When a map is printed with a starting line less than the current line, a form feed order is included in the first line of the print map before it is sent to its associated resource. The line counter is reset whenever a form feed order is included in the output.

When a printer map is printed with a starting line greater than the current line counter, VisualAge Generator inserts the needed carriage control characters in the print output, advancing the printer to the specified starting line.

Fixed position printer maps can be defined to start in a column other than column 1. However, only one map can be displayed on a given line. Side-by-side maps are not supported.

Floating Printer Maps

Like floating maps for display devices, printer maps do not have a fixed starting location. A floating area for the printer is defined in the same manner as for display devices. The floating area size and offset must not exceed the default device size. If a floating area is not defined for the printer, the floating area defaults to the full device size. Although floating printer maps cannot be defined with a specified starting column, the floating area can be defined to start in a column other than column 1.

When a floating printer map is printed, the map is positioned in the first available space in the defined floating area. If a program has previously displayed a fixed map, a portion of which falls in the floating area, a form feed order is inserted in the output and the floating map is placed in the first floating area position of the next page. If the floating map cannot fit at the bottom of the floating area, the floating map is placed in the first floating area position of the next page.

If different floating areas are defined for different print devices (PRINT, PRINT-B, 3767), only one of the floating area definitions is used for all the

printer maps. To avoid confusion, either specify the same print device for all print maps or specify the same floating area for all printer devices.

Printer paging control

Control characters are inserted into the print output to manage the occurrence of page advances or form feeds on the printer. Form feed orders are inserted as follows:

- When the program prints a fixed position map with a starting line number less than the current print line (as described in "Fixed position printer maps" on page 243).
- When a floating map is printed after the floating area has been partially overlaid by a fixed map.
- When a floating map is printed but there is not enough room remaining in the floating area for the entire map.
- When a map is printed that has previously been issued a SET map PAGE.
- When a CLOSE option is processed for any printer map. This inserts a form feed order before releasing the print output.
- When a main program ends.
- When a called program that was called by a non-VisualAge Generator program ends.
- When printing the next map after a CLOSE option is processed for a printer map.

Releasing print output

Print output is accumulated until the printer is "closed". The output is then released to the associated resource. The manner in which this accumulation and release are performed varies by environment and resource. When the printer is closed, a final form feed order is inserted into the output, advancing the printer to the top of the next page. There is no form feed when a program starts. The printer issues a form feed and closes the printer file when any of the following occurs:

- a program issues a CLOSE I/O option for a printer map.
- A main program ends.
- A called program that was called by a non-VisualAge Generator program ends.

  This enables you to create reports where a portion of the report (for example, the heading) is produced by the main program and other portions of the report (for example, sections of the body of the report) are produced by called programs.
- A segmentation break occurs in IMS/VS and CICS environments.

Note: For CICS OS/2, the following behavior occurs when using a VisualAge Generator program that does not have print maps, but which calls a

VisualAge Generator program that does have print maps. This is assuming that no explicit CLOSE is performed for the print map. If linktype=cicslink (the default) is specified at generation time, print maps will be closed on return from the called program. If the desired behavior is for print maps to be closed at the end of the run unit, then specify linktype=dynamic at generation time.

Performance techniques

The following are general considerations for generating COBOL programs that perform well:

- NUMC and PACK provide better performance than NUM and PACF.
- SET record EMPTY statements involving NUM or PACF fields that do not have substructures (ones that are not made up of other data items with high level numbers) run more slowly because they result in calls to VisualAge Generator Server for MVS, VSE, and VM (to set the sign bit).
- If the generation options /INITRECD and /INITADDWS are always specified, then it might not be necessary to explicitly define a SET record EMPTY statement at the beginning of the program.
- MOVE or MOVEA statements involving the following types of data run more slowly because they result in calls to VisualAge Generator Server for MVS, VSE, and VM:
  - Short HEX field to a longer HEX field (to handle padding of the target field with binary 0's)
  - MIX to MIX (to validate that the source is valid data and do the move) unless the /NOVALIDMIX generation option is specified.
  - Any move to a NUM or PACF field (to set the sign bit). A move to a NUM field in a CICS for OS/2 program is the only situation that does not result in a VisualAge Generator Server for MVS, VSE, and VM call.
  - CHA to HEX or HEX to CHA (to convert between the two data types and handle padding with binary 0's for HEX fields)
  - CHA to MIX or MIX to CHA (to convert between the two data types and handle SO/SI and padding/truncation).
  - On OS/400 runtime systems, NUM/PACKF or NUMC/PACK data types can be optimized such that the call to VisualAge Generator Server for AS/400 is avoided. See generation option /POSSIGN.
- IF or WHILE statements involving comparisons of the following types of data are *slow* because they result in calls to VisualAge Generator Server for MVS, VSE, and VM:
  - HEX to HEX where the fields are of different lengths (to handle padding of the shorter field with binary 0's)
  - CHA to HEX or HEX to CHA (to convert between the two data types and handle padding with binary 0's for HEX fields)

- CHA to NUM or NUM to CHA (to convert between the two data types).
- On OS/400 runtime systems, NUM/PACKF or NUMC/PACK data types can be optimized such that the call to VisualAge Generator Server for AS/400 is avoided. See generation option /POSSIGN.

In a RETR statement, similar *slow* comparisons occur when the search column and data item 1 differ as listed for IF and WHILE. This also occurs for FIND statements.

- Limit the number of maps in your map group and the number of device types specified.
- Use a separate map group for help maps. This splits the original map group format module into two separate modules. The map group format module for the help map only needs to be loaded when the program user requests help.
- A blank fill character for CHA, MIX, or DBCS map variables provides better performance than other fill characters.
- The choice of XFER or DXFR affects performance. See "Choosing between XFER and DXFR statements" on page 248 for detailed information.
- The choice of CALL or DXFR affects performance. See "Choosing between CALL and DXFR statements" on page 247 for more information.
- Generation options chosen can effect performance. Refer to the *Generation Guide* document for description of these options.
- If your terminal maps contain a large number of fields intended only for output (program never moves data from the field, tests the contents of the field, or sets the field unprotected or modified), define each output only field as protected on the map and specify /INEDIT=INONLY as a generation option when the program and map group are generated.
- For CICS and IMS/VS, when running in segmented mode, use a DL/I (IMS/VS) or main storage (CICS) work database for best performance. Refer to the *VisualAge Generator Server Guide for MVS, VSE, and VM* or the *VisualAge Generator Server Guide for MVS, VSE, and VM* document for general information.
- In the CICS environments, use EZECNVCM=1 as an alternative to running in segmented mode if storage resources are not constrained. Running nonsegmented with EZECNVCM=1 frees file and database resources across a CONVERSE but does not require the state of the program to be saved and restored in a temporary storage file.
- If a program is always to run in nonsegmented mode, do not reference the EZESEGM special function word in the program. Using EZESEGM causes extra logic to be generated into the COBOL module.

- Put the messages that are used most frequently at the top of the user message tables. Also, for CICS, define user message tables as **Shared** and **Resident** in the Table Definition facility. For IMS, preload user message tables.
- For both IMS and CICS, if you use the DXFR statement to transfer among segmented programs, use the EZESEGTR special function word to set the transaction id to a transaction associated with the program that is doing the CONVERSE, rather than returning to the initial program in the group.

Choosing between CALL and DXFR statements

The following considerations might help you decide whether a CALL or DXFR statement meets the needs of your program.

Passing parameters

A CALL can pass 30 parameters, including level-77 data items, maps and records.

A DXFR statement can pass only a single record.

Returning to the original program

A called program always returns to the calling program at the location in the calling program where the call was made.

A program transferred to with a DXFR statement can only return to the original program by using a DXFR or XFER statement. The original program starts from the beginning in this case.

Segmentation

A CALL must be to a called or called batch program. Segmentation is not supported in called programs. Any CONVERSE function in a called program must run in nonsegmented mode.

A DXFR must transfer to a main or main batch program. Segmentation is supported for main programs. A CONVERSE in a main program can run in either segmented or nonsegmented mode.

Linkage table

The type of CALL (static, dynamic, CICS LINK, or remote) can be specified in the linkage table. In addition, the format that parameters are passed in (OSLINK, CICSOSLINK, COMMPTR, or COMMDATA) can also be specified.

The type of DXFR (static, dynamic, OS XCTL, or CICS XCTL) can be specified in the linkage table. The format that parameters are passed in is determined by the type of DXFR being done.

Storage Considerations

For a CALL, both the main and called program must be in storage.

For a DXFR statement, if either OS XCTL or CICS XCTL is used, the storage for the original program is released.

PSB Structures

For MVS/TSO, MVS batch, IMS, and VSE batch, the same PSB must be used regardless of whether a CALL or DXFR statement is used.

For CICS, the same or different PSB can be used regardless of whether a CALL or DXFR statement is used.

DB2 plan

For all environments, the initial DB2 plan used by the called or transferred-to program is the same as that for the original program.

However, for segmented programs in CICS for MVS/ESA and for the IMS/VS environment, you can change the transaction name in EZESEGTR prior to a CONVERSE. This results in a different transaction being scheduled after the CONVERSE and, therefore, a different DB2 plan is used. You do not need to bind the original and transferred-to programs into the same DB2 plan if you use a DXFR and change EZESEGTR before the first CONVERSE and if the transferred-to program does not use any SQL functions prior to the first CONVERSE.

Commit points and freeing resources

In non-CICS environments, neither a CALL nor a DXFR cause a commit point.

In CICS environments, a CALL does not cause a commit point. A commit point occurs on a DXFR statement under the following conditions:

- If a transfer to a non-VisualAge Generator program occurs and a PSB is scheduled
- If /SYNCDXFR is specified and a PSB is scheduled
- If /NOSYNCDXFR is specified for the transferring program and the transferring program had scheduled a PSB and different PSB names were identified in the program specifications for the two programs.

Portability

If a called program does a CONVERSE, it cannot run in the IMS/VS environment. A CONVERSE in a called program must run in nonsegmented mode and each CONVERSE must run in segmented mode for the IMS/VS environment.

Choosing between XFER and DXFR statements

When you transfer control between programs, you can use either an XFER or a DXFR statement. The following considerations might help you decide which statement meets the needs of your program.

Number of transactions

XFER forces the transaction code to change.

DXFR requires that the transaction code stay the same. All programs that transfer among themselves using a DXFR are part of the same run unit.

PSB structures

For CICS, the transferred-to and -from program can use the same or different PSBs for transfers using DXFR or XFER.

For MVS/TSO, MVS batch, IMS BMP, and VSE batch, the same DL/I PSB is required for both the transferred-from and -to programs for transfers using DXFR or XFER.

For IMS/VS, XFER requires each program to use a different IMS PSB. However, the two PSBs can use the same set of PCBs. In this case, the PSB specified in the program specifications can be the same.

For IMS/VS, DXFR requires that the PSB stay the same. All programs that transfer among themselves using a DXFR are part of the same run unit and must use the same PSB structure.

DB2 plan

For CICS for MVS/ESA and IMS/VS, XFER starts a new transaction.

CICS for MVS/ESA enables you to use the same plan for multiple transactions. However, if you are using the XFER statement to transfer from one program to another, it is not necessary to bind the programs into a single plan.

For IMS/VS, the plan name, program name, and the IMS PSB name must all be the same name. In this case, if you are doing an XFER between two programs, you must use two different plan names. A program transferring control to itself with an XFER statement is the only situation where the plan name remains the same.

For CICS for MVS/ESA and IMS/VS, DXFR requires that the programs use the same DB2 plan and that the DBRMs be bound together.

For MVS/TSO, MVS batch and IMS BMP, the same DB2 plan is required for both the transferred-from and -to programs for transfers using DXFR or XFER statements.

Security

The differences in transaction codes, PSBs, and DB2 plans have an effect on security.

Performance tuning

The differences in transaction codes, PSBs, and DB2 plans have an effect on performance tuning.

A DXFR statement should give better performance than an XFER statement. With the DXFR statement a transaction does not end when transferring control to another program.

For CICS, if an XFER statement is necessary, you should consider using the /GENRET generation option. This option causes the XFER statement to be generated into a CICS RETURN IMMEDIATE command instead of a CICS START command. This method uses less CICS overhead to transfer control to a new transaction.

Transaction scheduling

XFER causes CICS or IMS/VS to schedule a new transaction. If XFER without a map is used, the transaction is immediately available for scheduling. If XFER with a map is used, the transaction is not scheduled until the program user enters data.

DXFR does not cause CICS or IMS/VS to schedule a new transaction.

Commit points and freeing resources

For CICS and IMS/VS, XFER causes a commit point. Database locks are released. For CICS, the program control logic returns to CICS. For IMS/VS, the program control logic reads the next message on the message queue.

For MVS/TSO, MVS batch, VM CMS, VM batch, and batch-oriented IMS BMPs, a commit point occurs for an XFER statement if the /SYNCXFER generation option was specified. For MVS/TSO and VM CMS, a commit point also occurs for an XFER statement in a main transaction program that is defined as segmented or single-segment. For a transaction-oriented IMS BMP program, an XFER statement does not cause a commit point.

For non-CICS environments, DXFR does not cause a commit point. Database locks continue to be held. The control program handles the transfer and processing continues until the transferred-to program does a CONVERSE, XFER, or EZECLOS.

For CICS, a SYNCPOINT occurs on a DXFR under the following conditions:

- If a transfer to a non-VisualAge Generator program occurs and a PSB is scheduled
- If /SYNCDXFR is specified and a PSB is scheduled
- If /NOSYNCDXFR is specified for the transferring program, the transferring program had scheduled a PSB, and different PSB names were specified in the program specifications for the two programs.

Loading modules

For CICS, XFER causes the transferred-to program to be loaded, unless it is already loaded.

For CICS, DXFR causes the transferred-to program to be loaded, unless it is already loaded. When a CONVERSE occurs in a transferred-to module, two loads might be required when the program user enters data, one for the module associated with the transaction code and one for the transferred-to module.

For IMS/VS, XFER causes the transferred-to program to be loaded unless it is preloaded or there is a message already on the input message queue for that transaction.

For IMS/VS, DXFR using a dynamic call causes the transferred-to program to be loaded unless it is preloaded or it has already been used during this scheduling of the from transaction by IMS (considering everything that has gone on for all loop backs to read the next message off the queue). When a CONVERSE occurs in a transferred-to module, two loads might be required when the program user enters data, one for the module associated with the transaction code and one for the transferred-to module.

For IMS/VS, DXFR using a static call does not result in a load because the programs are already linked together.

Using First Map

XFER with or without a map to a program that has a First Map is permitted. For IMS/VS, XFER with a map requires the transferred-to and -from programs to use the same map group.

DXFR with a map is not permitted. DXFR to a program that has a First Map is not permitted.

XFER with a map requires the transferring and transferred-to programs to share the map. For CICS, MVS/TSO, and VM CMS, you can use different map groups when using XFER with a map only if you copy the map definition into the second map group. For IMS/VS, XFER with a map requires the transferred-to and -from programs to use the same map group.

The DXFR statement allows the two programs to use different map groups.

Work database storage

Use the /WORKDB generation option to specify the type of work database you want to use. DL/I and SQL are supported for IMS/VS. Main or auxiliary temporary storage are supported for CICS. For IMS/VS, XFER causes, at most, the record and map to be stored in the work database. For CICS, the record is stored in the COMMAREA and the map is stored in the work database.

The DXFR statement does not use the work database. When you use a CONVERSE function, program context, EZE special function words,

all records and all maps that the program uses are stored in the work database. If you choose to use the XFER statement or DXFR statement for segmented programs then the there are no special storage consideration when a CONVERSE I/O option is performed. If you choose to use the single-segment mode with segmented programs, then the storage saved at the CONVERSE I/O option can be significant.

If the transferred-to program immediately performs a CONVERSE I/O option, then the XFER statement performs slower than a DXFR statement. The slower performance is because of scheduling done by IMS. Using an XFER statement with a map to a program with a First Map should perform faster than a DXFR statement to a program that immediately issues a CONVERSE I/O option. In either situation, IMS schedules a transaction. The XFER statement minimizes the amount of information sent to the work database.

Productivity

The CONVERSE I/O option is easier to define than single-segment mode, especially for developers already familiar with VisualAge Generator.

Cross-Platform development between OS/2 and Windows NT

Because of the code page differences between OS/2 and Windows NT, VisualAge Generator does not support cross-platform development between OS/2 and Windows NT. The areas of VAGen parts that might contain character data with different codepoints for the OS/2 and Windows platforms are:

- Part names containing national characters
- Literals in logic statements
- The alternate ″not sign″ (the sign used on the host systems) in logic statements
- Map fields (constant fields and initial values of variable fields)
- Table contents
- Labels and initial contents in views (GUIs)
- Comments in logic parts, descriptions and prologs

If you choose to cross-develop between OS/2 and Windows NT, it is your responsibility to ensure that only character data with the same codepoints are used on both platforms. If characters with different codepoints are used in a part developed on one platform, when you load this part on the other platform, the characters will appear incorrectly.

In general, you should be able to cross-develop for both OS/2 and Windows NT if you use only the following character data:

- ASCII characters (codepoints less than 128)

- DBCS characters
- The caret (″^″) instead of the alternate ″not sign″

The strings used in GUI views can be extracted into separate files. These files contain information about the platform from which they were extracted. When these files are loaded on another platform, the correct codepoint conversion is performed for the strings. Refer to “Developing multi-language GUI clients” document for more information. Note that this process enables you to test and run GUI views correctly but it does not help in editing the GUI clients. This means that when you edit a GUI view on Windows NT that was defined in OS/2, you might see invalid characters in your strings.

Developing multi-language programs

If you want to use multiple languages in a single CICS region or IMS system, you must change the program name and assign separate transaction codes to each version of the program. For the programs to co-exist in the same CICS region or IMS system, each must have a unique name. You must also rename map groups and tables if they contain NLS dependent information. Any references to these map groups and tables must also be changed to the new name.

For CICS, you must have PPT and PCT entries for each version of the program.

For IMS, you must have APPLCTN and TRANSACT macros in the IMS system generation for each version of the program.

If you want to use multiple languages under MVS/TSO, you do not need to change the program, map group, or table names, provided you install generated programs into separate load libraries. You must use the correct load library for the language you want to run.

Developing multi-language GUI clients

If you are developing GUI clients to run with multiple languages, or if you want to deploy your clients on multiple platforms, VisualAge enables you to extract the text of your user interface into separate files. This extraction enables you to translate the text strings. It also enables VisualAge to perform the correct codepoint conversion when the strings are read in on a different platform than the one on which they were extracted. Following is a summary of the steps needed to create an external file containing all of the text strings used in the visual parts of an ENVY package/application:

1. Define a binary file to hold the text strings. The file must have an extension of .mpr.

2. Register the binary file when the package/application is loaded
3. Unregister the binary file when the package/application is unloaded
4. Build the binary file
5. Bind the image strings

Note: This section only applies to VisualAge Generator Smalltalk users.

For a detailed discussion of this process, refer to the "National Language Support" section of the *VisualAge Smalltalk User's Guide* or the VisualAge for Java online VisualAge Generator help facility.

Coding arithmetic operations for consistent math results

Due to differences in COBOL, Smalltalk, C++, and Java arithmetic operations involving multiplication, division, and remainders can have different results in different operating environments when decimals places are greater than zero. To ensure consistent math results in all VisualAge Generator operating environments, you should code each arithmetic operation as one statement, and break up remainder operations into their component parts.

Coding multiplication and division operations

To get consistent results from multiplication and division, code each operation separately, as follows:

```
intermediate_result = a * b;
result = intermediate_result / c;
```

Do not code an arithmetic operation like the following:

```
result = a * b / c;
```

The number of decimal places defined to the `intermediate_result` item determines the accuracy of the final result.

Coding division operations for consistent remainders

To get consistent remainders from division operations, code each operation separately, as follows:

```
quotient = dividend / divisor;
remainder = dividend - (quotient * divisor);
```

Use a quotient with 0 decimal places to get Smalltalk, C++, or Java type remainders. Use a quotient with the same number of decimal places as the remainder to get COBOL type remainders.

The remainder operator provides consistent results if the dividend, divisor, and remainder fields are all defined with 0 decimal places.

Appendix B. Naming and programming standards

This appendix provides an example of naming and programming standards you might use in your development group. Each computer installation should have a set of programming standards and guidelines to ensure consistency and ease of operation among application systems.

Suggested naming conventions

VisualAge Generator Developer permits shared access to several ENVY packages/applications during the development process, either by one developer or concurrently by many. The names of the various parts should be unique across all ENVY packages/applications used during development of a specific system.

The use of a well-designed, enforced naming scheme is essential to the management and maintenance of a successful development environment.

This section includes the following information:
- Example naming standards
- Guidelines for programming standards

Repository/ENVY library part names

Your ENVY packages/applications should have a unique prefix (possibly 3 characters) so that all of your packages/applications will be grouped alphabetically in the VisualAge Organizer/VisualAge for Java Workbench. Using a single prefix for your packages/applications means that all of your packages/applications appear together on the VisualAge Organizer/VisualAge for Java Workbench. If there are several subsystems, each subsystem could have a different prefix, but this means that they might not appear together on the VisualAge Organizer/VisualAge for Java Workbench.

For example, if you have an Accounting system and a Payroll system, you might choose to prefix your ENVY packages/applications with *Acct* for the Accounting system and *Pay* for the Payroll system. However, there are also sample packages/applications that ship with VisualAge Generator. These are prefixed with *Hpt*. If you have the packages/applications for accounting, payroll, and the VisualAge Generator sample packages/applications all loaded into your workspace/image at the same time, the order in which the packages/applications appear on the VisualAge Organizer/VisualAge for Java Workbench is:

```
Acct
Hpt
Pay
```

Therefore, you might prefer to use a naming convention such as *XyzAcct* for the Accounting system and *XyzPay* for the Payroll system, where *Xyz* is an acronym for your company name. In this case, the order in which the packages/applications appear on the VisualAge Organizer/VisualAge for Java Workbench is:

```
Hpt
XyzAcct
XyzPay
```

Versions in ENVY should also have a naming convention. Using the default naming convention (1.0, 1.1, and so on) might be the best way to start.

Repository/ENVY library part names

With the exception of nonshared data items within a record, VisualAge Generator part names must be unique within an ENVY package/application regardless of the part type. For example, a program cannot have the same name as the record that defines the working storage used by that program. Also, if two projects are part of the same package/application system, the names of the parts in the two projects must not overlap. Prefix or suffix characters can be used to partition names by system, program, and part type to avoid overlapping names for different parts.

For example, the format for a part name is *sspptnn...n*

Where:

ss Two alphabetic characters designating the application system ID in which the part is used(an application system is a group of programs designed to do a function, such as payroll). You might reserve the characters ZZ for parts common to more than one application system.

pp Two numeric digits identifying parts associated with a specific program in an application system. 99 might be reserved for parts of the same application system common to more than one program in the system. The developer assigns the program numbers.

Note: The combination ZZ99 is then reserved for skeleton parts used as a development base for all application systems.

t(t) One or two alphabetic characters designating one of the following types of parts:

A Program
D Dummy record for extra records list
F File

P Function
G Map group
M Map
PS PSB
R Record
T Table
W Working storage
X Alternate index record

nn...n The developer assigns a maximum number of characters permitted for that part. Always start with 01, rather than 00.

Example

VA01P01-DISP-ACCT is a function and VA01W is working storage related to program VA01A.

Data item names

Data item names can be from 1 to 32 characters long, but it is suggested that they be at least 2 characters and no more than 30 characters. The 30-character limitation is to avoid alias names being assigned for generated COBOL, and the 30-character limitation to improve readability of the program. The following naming format can be used for data items:

abbbbbbb...b

Where:

a An alphabetic first character. The letter Z in this position indicates a data name common to all application systems.

bbbbbbb
Up to 31 alphanumeric characters

Example

UCOMMAND is the name of the user command entry variable field.

To avoid aliases being assigned during COBOL or C++ generation and to improve the readability of the generated COBOL or C++ program, follow these standards:

- Use 30 characters or less in item names.
- Do not use COBOL or C++ reserved words.
- Do not use $, #, @, or _ characters.
- Do not use double-byte character set (DBCS) names for item names if your program contains SQL functions.
- For C++ programs, do not use DBCS names.

Suggested programming standards

This section outlines suggestions for developing your own set of programming standards. It is important to develop a set of standards within your development group to ensure consistency between programs.

Function Standards

Use invoked functions for I/O functions that need to run multiple times or from multiple places. Test for any PA key after invoking CONVERSE. Write your program so that if a PA key is pressed the program does not accept the input as valid. Make the program go to an exit, or display the function again.

Note: IMS/VS reserves the PA key so you cannot use it.

Map Standards

Include a line for EZEMSG.

Include a map identifier on the map, which could also identify the version of the map.

Make use of the user help map capability and have the VisualAge Generator product display the help maps for you. This method is more efficient than defining VisualAge Generator logic statements to check for specific user responses and then using a CONVERSE function to display the help map. Using the VisualAge Generator product also eliminates the need to know exactly what (and how many) help maps there are when defining programs. The help maps can be defined during map definition and then ignored in the logic of your program.

Appendix C. Size restrictions and record lengths

Size limitations for VisualAge Generator

Table 27 outlines size limitations for VisualAge Generator. Refer to specific language element compatibility considerations for additional environmental restrictions.

Table 27. Size Limitations for VisualAge Generator

Definition	Limitations
Number of Data Items	32767 data items and literals per program
Data Items	32767 bytes in record definition 32730 bytes in record definition (OS/400 only) 254 bytes in table definition 8000 bytes for printer maps (IMS only) 1 byte less than map size for terminal maps (IMS only)
Map Constant Field	255 bytes (IMS only)
Working Storage	32767 bytes if used in an XFER or DXFR statement 32730 bytes maximum, regardless of XFER or DXFR (OS/400 only)
Numeric Items	18 digits
Decimal Places	18 digits (included within numeric item size)
Subscripting	One level
Number of Occurrences	32767 in record definition 32730 in record definition (OS/400 only)
Maximum Table	524288 bytes for MVS, VM, VSE, or non-shared tables on CICS for OS/2 64K bytes for shared tables on CICS for OS/2 On OS/400, table rows are limited to 32,767 bytes, total table contents is limited to 3 mega bytes.
Maximum Number of Variable Fields on a Map	800 on CICS for OS/2
Primary Table Columns	700 top level data items
Numeric Literals	18 digits plus 1 sign, 1 decimal point, or both

Table 27. Size Limitations for VisualAge Generator (continued)

Definition	Limitations
CALL Parameters	Limit of 30 arguments
Number of Main Functions	254 per program
COMPILED Size of a Function	64K limit for COBOL for the Micro Focus COBOL compiler on OS/2
Number of lines in an SQL statement	819

Maximum record lengths

Table 28 outlines the maximum record lengths for each environment.

Table 28. Maximum Record Lengths by Environment When using an XFER Statement

Environment	XFER with Record	XFER with Record and Map
CICS (Main or Main Batch)	32763 (limit set by CICS)	32753 (Main only) - (32763 - 10 bytes reserved for VisualAge Generator Server for MVS, VSE, and VM, and VisualAge Generator Server for AIX, Windows NT, and Solaris)
MVS/TSO (Main or Main Batch)	32767	32767 (Main only)
MVS batch (Main Batch)	32767	XFER not supported
IMS/VS (Main)	32753	32753
IMS/VS (Main Batch)	XFER not supported	XFER not supported
IMS BMP (Main Batch)	32767	XFER not supported
OS/400 (Main or Main Batch)	32730	32730
VM CMS	32767	32767
OS/2 batch	32767	XFER not supported
AIX batch	32767	XFER not supported
Windows NT batch	32767	XFER not supported
HP-UX batch	32767	XFER not supported
Solaris batch	32767	XFER not supported

Table 29. Maximum Record Lengths by Environment When Using a DXFR Statement

Environment	DXFR with Record
CICS	32763
MVS/TSO	32767
MVS batch	32767
IMS/VS	32767
IMS BMP	32767
OS/400	32730
VSE	32767
VM CMS	32767
VM batch	32767
OS/2 batch	32767
AIX batch	32767
Windows NT batch	32767
HP-UX batch	32767
Solaris batch	32767

Table 30. Maximum Record Lengths for Serial, Relative, and Indexed Records by Environment

Environment	Serial, Relative, and Indexed Records
CICS (VSAM)	32763 (32688 for journaled records) 9999 for serial and indexed records on CICS for OS/2 4092 for relative records on CICS for OS/2
CICS (TD queue)	32763 (9999 on CICS for OS/2)
CICS (TS queue)	32762
CICS (Spool)	32763
IMS/VS (message queue)	32755 (32767 - 2 bytes for LL, 2 bytes for ZZ, and 8 bytes for transaction name) (IMS/VS Main can only ADD to a message queue)
IMS BMP (Main batch)	32755 (32767 - 2 bytes for LL, 2 bytes for ZZ, and 8 bytes for transaction name)
OS/400	32730

Table 31. The Maximum Audit Data Length for a Record by Environment

Environment	Audit Data Length
CICS	32763
MVS batch (with DL/I)	32765 (32767 - 2 bytes ZZ)
IMS/VS (with DL/I)	32765 (32767 - 2 bytes ZZ)
IMS BMP (with DL/I)	32765 (32767 - 2 bytes ZZ)
VSE, VM	not supported

Table 32. The Maximum CREATX Data Length for a Record by Environment

Environment	CREATX Data Length
CICS	32763
IMS/VS	32765 (32767 - 2 bytes ZZ)
IMS BMP	32765 (32767 - 2 bytes ZZ)

Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Readers' Comments — We'd Like to Hear from You

VisualAge Generator
Design Guide
Version 4.0

Publication No. SH23-0264-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SH23-0264-00

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

IBM Corporation
Information Development
Department G7IA / Bldg 062
P.O. Box 12195
Research Triangle Park, NC
27709-2195

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

SH23-0264-00

Cut or Fold
Along Line

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SH23-0264-00