VisualAge Generator

External Source Format Reference

Version 4.5

SH23-0265-01

Note

Before using this document, read the general information under "Notices" on page vii.

First Edition (September 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:

- AD/Cycle
- AIX
- AS/400
- CICS
- CICS OS/2
- CICS/ESA
- CICS/MVS
- CICS/VSE
- CICS/6000
- COBOL/370
- COBOL/400
- DB2
- IBM
- IMS
- IMS/ESA
- Language Environment
- MVS
- MVS/ESA
- Operating System/2
- OS/2
- OS/400
- SAA
- SQL/DS
- SQL/400
- System/370
- VisualAge
- VisualGen
- VM/ESA

The following are trademarks of other companies:

HP-UX	Hewlett-Packard Company
Micro Focus	Micro Focus Limited

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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 is a reference for the VisualAge Generator Developer external source format. This document describes the external source format syntax and the VisualAge Generator structures only as they relate to the external source format. For additional information on VisualAge Generator structures, refer to the VisualAge Generator help facility.

Who should use this reference

This document is intended for application developers who want to convert applications developed with a non-VisualAge Generator design tool into VisualAge Generator Developer programs or who want to analyze and modify the VisualAge Generator source code. It also provides application developers a method to export VisualAge Generator definitions to other systems.

For information on defining programs on a workstation using the VisualAge Generator Developer product, refer to the VisualAge Generator library and the VisualAge Generator help facility.

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]

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.

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]

4. This document is included when you order the VisualAge Generator Server product CD.

Chapter 1. Introduction

The external source format feature of VisualAge Generator lets you define VisualAge Generator parts in a readable format.

External source format provides the following capabilities to make program development easier:

- Accepts program definitions into VisualAge Generator that were developed using non-VisualAge Generator design tools
- Lets you export VisualAge Generator definitions to other systems
- Provides readable program definitions within VisualAge Generator
- Provides an external format of VisualAge Generator definitions that can be used for centralized control and archiving
- Lets you access and analyze VisualAge Generator source code.

When you export a VisualAge Generator part, the part is saved in external source format. See Appendix A. External source format functions with VisualAge Generator commands, for information on using the external source format with VisualAge Generator.

External source format tags

The external source format uses a set of mark-up language elements, called tags, to define each of the parts found in a library. These tags identify the type of part and the attributes and values assigned to that part. A unique tag delimits each part type.

An example of the tag syntax follows:

```
:tagname
keyword=value.
tag-content
:etagname.
```

Tag name

The *tagname* specifies the name of the tag. Tag names are prefixed with the start tag open delimiter, the colon (:), which must be in column 1. Tags can be specified in uppercase and lowercase. Examples of tag names used in external source format are: MAP, ITEM, VFIELD, and PSB. Some tags require that you mark the end of an element explicitly. Ending tags consist of the tag name prefixed with the end tag open delimiter (:e). Both start and end tags close with the tag's closing delimiter, the period (.).

Note: The tag's closing delimiter is only required when you have text following the tag name, but it is a good practice to always include it.

An external source format tag in column 1 marks the end of the preceding tag and the beginning of the next tag.

Tag attribute

The second item in the example shown in External source format tags, *keyword=value*, is an attribute of the tag. In this example, *keyword* is the name of the attribute and *value* is the variable attribute data. In some cases, the attribute value can be a list of numbers or alphabetic characters separated by one or more blanks.

You must enclose attribute values within single quotation marks or double quotation marks when the value contains a blank. If you need quotation marks inside the value, you must use either two single quotation marks or two double quotation marks, as in the following example:

```
:tagname keyword='value containing blanks, "single quoted text," and
""double quoted text.""'.
```

When an attribute value is a number that contains a decimal point, place at least one digit to the right of the decimal point to distinguish it from the tag's closing delimiter.

An attribute and its value can span multiple lines. Multiple attributes appearing on the same line must be delimited by one or more blanks. Attributes can be specified in lowercase and uppercase.

Tag text

The *tag-content* is the text associated with that tag. The tag-content, which is subject to national language translation, follows the period. The format shown in "Format example of variable field definition" on page 4 is used throughout this publication to describe the tag layouts.

The following information is provided for each tag.

Name
The name of the tag.

Description
A short description including all the required and optional tags.

Syntax
The tag syntax always appears in the left hand column. The tag being defined is always listed in bold typeface.

Note: You must specify tags in the order they are listed under "Syntax." However, you do not have to specify the attributes in the order they are listed under "Attributes."

Attribute table

The table to the right of the syntax lists all the required and optional attributes and values for the current tag. The tables list attributes in uppercase letters. You can specify attributes in external source format in uppercase and lowercase. These tables have three columns:

Attributes
Lists all possible attributes

Values
Lists all possible attribute values

The tables use several styles of highlighting to make the values more readable:

- *Italic* — Italic typeface indicates a value name

  When you specify this attribute, you give this value a quantity particular to the part you are defining.
- **Bold** — Bold typeface indicates a default value

  If you do not specify this attribute, VisualAge Generator automatically uses this value.
- CAPS — Words in all capital letters indicate the choices you can specify for this attribute

Uses Lists possible uses

Brackets ([]) surround attributes and values that are optional for the current tag. Vertical bars (|) separate value options that are valid for an attribute. Braces ({}) surround attribute value options that are mutually exclusive (you can specify only one of them). Single quotation marks (' ') surround attribute value options when the value contains a blank or special character.

Some tags do not require keyword attributes. Tags such as PROL, JOINCON, and STMTS have value content only. The syntax tables for these tags have only two columns, **Value** and **Usage**.

Attribute description

A brief attribute description follows the syntax. This description contains two kinds of information:

- Quick reference
- Usage

Attribute quick reference: The attribute quick reference information appears inside labeled boxes. The boxes list all the attribute values, using the same highlighting that appears in the attribute tables:

- *Italic* indicates a value name.
- The phrase **(default value)** indicates a default value.
- CAPS indicate a value choice.

Attribute usage: A description of how to use an attribute follows the quick reference. In some cases, you might require a more detailed explanation of how to use the attribute. Refer to the VisualAge Generator help facility for more information about how to use the attributes.

Format example of variable field definition

The VFIELD tag specifies information about a variable field on the map. The optional tags MAPEDITS, MESSAGES, and VATTR tags provide additional information about a variable field. The EVFIELD tag closes the definition and is required for each VFIELD.

VFIELD tag attributes

The VFIELD tag specifies information about a variable field on the map. The optional tags MAPEDITS, MESSAGES, and VATTR provide additional information about a variable field. The EVFIELD tag closes the definition and is required for each VFIELD.

Syntax	**Attributes**		
:map . . .	***:VFIELD***		
:vfield . . .	**Attributes**	**Values**	**Uses**
	BYTES=	*field length in bytes*	Specifies the number of positions the variable field occupies
:mapedits	COLUMN=	*column number*	Specifies the column of the byte immediately preceding the variable field
. . .	[DECIMALS=	{**0** | *decimal places*}]	Specifies the number of positions to the right of a decimal point in numeric items
:messages	[DESC=	*'field description'*]	Describes what the field represents
. . .	[EDITORDR=	*number*]	Specifies the map edit sequence
:vattr	[INDEX=	*index value*]	Specifies the index value when the field is an array

Syntax	Attributes		
. . .	[NAME=	*field name*]	Identifies a map variable field
:evfield	ROW=	*row number*	Specifies the row of the byte immediately preceding the variable field
. . .	[TYPE=	{**CHA** | DBCS | MIX | NUM}]	Specifies the data type
:emap	[**.**[*initial value*]]		Specifies the initial value of the field
	:EVFIELD [.]		

BYTES=

field length in bytes
Specifies the number of positions the variable field occupies

COLUMN=

column number
Specifies the column of the byte immediately preceding the variable field

DECIMALS=

0 Specifies 0 decimal places (default value)

decimal places
Specifies a number of decimal places

DECIMALS are the number of positions to the right of an implied decimal point in numeric items. Decimal places can be specified only for numeric data. The maximum number of decimal positions is 18 or the number of digits defined for the field, whichever is smaller.

DESC=

'field description'
Describes what the variable field represents

DESC is a text description of what the variable field represents. The text can be up to 30 characters.

EDITORDR=

number
Specifies the map edit sequence

EDITORDR allows you to modify the editing sequence in which variable fields on the map are checked on input to a program.

Chapter 2. Program structures

This chapter describes the external source format tags and attributes that define a VisualAge Generator program. See "Chapter 3. GUI client structures" on page 21 for information about Graphical User Interface (GUI) clients.

Program definition

The PROGRAM tag defines attributes for individual programs. The PROGRAM tag defines a VisualAge Generator program that consists of various VisualAge Generator objects, the parameter list of the program, the type of program, and text describing the program. The following tags are used with the PROGRAM tag to further define aspects of a program. They are discussed later in this chapter.

- MAINFUN
- TABREC
- CALLPARM
- PROL

All these tags can have attributes. The EPROGRAM tag closes the definition and is required.

Program and table generation option specification

The GENOPTS tag and the GENTABLE tag specify the saved generation options associated with the program. These tags and their associated attributes are supported to provide compatibility with Cross System Product and to help you migrate programs to VisualAge Generator.

To migrate Cross System Product generation options to VisualAge Generator, use the external source format conversion utility to extract the generation options. For information on this utility, refer to the *Migrating Cross System Product Applications to VisualAge Generator* document.

:PROGRAM tag attributes

Syntax	**Attributes**		
:program . . .			
	:PROGRAM		
:mainfun . . .	**Attributes** [BYPKEY=	**Values** *number list*]	**Uses** Specifies keys used to bypass map edits and map edit groups

Syntax	Attributes		
:emainfun	[DATE=	'*modification date*']	Specifies the date the program part was last modified
:	[EXECMODE=	{**NONSEGMENTED** | SEGMENTED | SINGLESEG}]	Specifies the execution mode
:tabrec	[FIRSTMAP=	*map name*]	Specifies the initial map to be shown prior to running the first function
	[FIRSTUI=	*user interface record name*]	Specifies the name of the record that is to contain data to be used by the bean
:	[HELPGRP=	*help map group name*]	Specifies the map group that contains user-defined help maps
:callparm	[HELPKEY=	*number*]	Specifies a key to use for requesting help
:	[IMPLICIT=	{**Y** | N}]	Specifies whether generation and test are to create implicit data items
:prol	[MAPGROUP=	*main map group name*]	Specifies the map group that contains the first map for the program, or the maps to be specified as I/O objects, or as received parameters in the called parameter list
:	[MSGTABLE=	*message table prefix*]	Specifies the message table name prefix
:eprol	NAME=	*name*	Identifies the program being developed
:vagt	[PFEQUATE=	{**Y** | N}]	Makes pressing F13-24 equal to pressing F1-12
:	[PSB=	*PSB name*]	Specifies a program specification block definition in the library
:tracbag	[TIME=	'*modification time*']	Specifies the time the program part was last modified
:	[TYPE=	**{MAIN** | MAINBATCH | CALLED | CALLBATCH | WEBMAIN}]	Defines the method of processing to be used for a program

Syntax	Attributes		
:etracbag	[WORKSTOR=	*record name*]	Specifies a record with working storage organization
	[.]		
:eprogram	***:EPROGRAM*** [.]		

BYPKEY=

number list
Specifies up to five keys that allow the user to bypass map edits and map edit groups

Specify BYPKEY keys as integers from 1 to 24. Separate multiple keys with blanks. No default bypass edit keys are designated.

Note: Specifying the bypass edit keys for a map overrides the program specification bypass keys for that map. The bypass edit keys cannot be the same as the help key.

DATE=

'modification date'
Specifies the date the program part was last modified

DATE format is mm/dd/yyyy. Single quotation marks are optional.

EXECMODE=

NONSEGMENTED
Specifies the program runs in conversational mode (default value)

SEGMENTED
Specifies that the program runs in CICS pseudoconversational mode, IMS conversational mode, or IMS nonconversational mode

SINGLESEG
Specifies the program runs in single-segment mode and is valid for IMS/VS only

EXECMODE defines the mode in which programs run in the transactional processing environments. Use the EZESEGM special function word in the program to dynamically change the execution mode during execution.

NONSEGMENTED holds input/output locks, position, and main storage resources across each CONVERSE I/O option.

SEGMENTED saves program information across each CONVERSE I/O option. All main storage resources are freed while waiting for a terminal. SEGMENTED is valid only when the program is a main transaction.

SINGLESEG specifies that the program runs in single-segment mode. It is valid for IMS/VS only. A single-segment program processes a single input from a terminal and ends after it responds to the input or after it transfers to another program. Program information is not saved when the map is displayed.

FIRSTMAP=

map name
Specifies the initial map to be shown prior to running the first function

FIRSTMAP is the initial map on which you enter data prior to running the first function. FIRSTMAP is valid only when the program is a main transaction.

FIRSTUI=

user interface record name
Specifies the name of the record that is to contain data to be used by the bean

FIRSTUI is valid only when the program is a WEB MAIN transaction.

HELPGRP=

help map group name
Specifies the map group that contains the user-defined help maps

HELPGRP can be the same map group as the one specified in the map group attribute. If so, the help map group attribute does not need to be specified.

Map group names cannot contain the characters $, #, or @.

HELPKEY=

number
Specifies a key to use for requesting help from program maps

Specify HELPKEY as an integer ranging from 1 to 24.

Note: The help key cannot be the same as any of the bypass edit keys.

IMPLICIT=

Y Creates implicits for all unqualified data items used in the program that are not defined in any map, record, or table used by the program (default value)

N Specifies that all processing involved in creating implicits is bypassed

IMPLICIT specifies whether implicit data item definitions are created. If IMPLICIT=N, test and generation issue error messages for any undefined items.

MAPGROUP=

main map group name
Specifies the map group that contains the first map for the program, or the maps to be specified as I/O objects, or as received parameters in the called parameter list

MAPGROUP is the name of the map group that contains the first map for the program, or the maps to be specified as I/O objects, or as received parameters in the called parameter list. A called program can use a different map group than the calling program unless a map is a parameter passed to the called program. In this case, the same map group is used.

Map group names cannot contain the characters $, #, or @.

MSGTABLE=

message table prefix
Specifies the message table name prefix

MSGTABLE is the prefix that links the message table to the program. The format of the message table name is XXXXyyy, where XXXX is the value of the message table prefix, and yyy is the NLS code. MSGTABLE consists of 3 to 4 characters and should meet the following conventions:

- The first character is alphabetic (A-Z)
- The remaining characters are alphanumeric (A-Z, 0-9)
- The name cannot contain blanks or have an EZE prefix.

NAME=

name Identifies the program being developed

NAME identifies the program being developed. It consists of 1 to 7 characters and must meet the following conventions:

- The first character is alphabetic (A-Z)
- The remaining characters are alphanumeric (A-Z, 0-9)
- The name cannot contain blanks or have an EZE prefix.

PFEQUATE=

Y Makes F13-24 equal to F1-12 (default value)

N Makes F13-24 not equal to F1-12

PFEQUATE makes pressing F13-24 equal to F1-12. If PFEQUATE=Y, you can code key checks into the program as if there are only 12 keys. Statements checking for a single key, such as F3, are treated as true if either F3 or F15 is pressed.

PSB=

PSB name
Specifies a program specification block definition in the library

The PSB is used to generate default DL/I calls within the program.

TIME=

'modification time'
Specifies the time the program part was last modified

TIME format is hh:mm:ss. Single quotation marks are optional.

TYPE=

MAIN Specifies main transaction processing (default value)

MAINBATCH
Specifies main batch processing

CALLED
Specifies called transaction processing

CALLBATCH
Specifies called batch processing

WEBMAIN
Specifies main transaction processing with a WEB browser

TYPE defines the method of processing to be used for a program:

- **MAIN**

  Specify MAIN for a program that interacts with a user at a display device. The program is started through a transfer from the system, a non-VisualAge Generator program, or a VisualAge Generator program. A block of working storage data can be passed to the program upon transfer from another non-VisualAge Generator program or VisualAge Generator program.
- **MAINBATCH**

  Specify MAINBATCH for a program that processes without interacting with a user at a display device. The program is started through a transfer from the system, a non-VisualAge Generator program, or a VisualAge Generator program. A block of working storage data can be passed to the program on transfer from another non-VisualAge Generator program or VisualAge Generator program.
- **CALLED**

  Specify CALLED for a program that is called by and returns to another VisualAge Generator program or non-VisualAge Generator program. The

program interacts with a user at a display device. Parameters can be passed between the calling and the called VisualAge Generator programs or non-VisualAge Generator programs.

- **CALLBATCH**

  Specify CALLBATCH for a program that is called by and returns to another VisualAge Generator program or non-VisualAge Generator program. The program does not interact with a user at a display device. Parameters can be passed between the calling and called VisualAge Generator programs or non-VisualAge Generator programs.

- **WEBMAIN**

  Specify WEBMAIN for a program that interacts with a user at a display device when the user interface is designed with a WEB Browser.

WORKSTOR=

record name
Specifies a record with working storage organization

WORKSTOR defines a temporary data area, in addition to the defined maps and records, that the program might need for processing. You can specify only one record for WORKSTOR, but you can specify additional working storage records in the Table and Additional Records List.

Main function definition

The MAINFUN tag is an optional tag that, with its attributes, further describes the PROGRAM and EPROGRAM tag set. If repeated, the order the functions are named determines the order they appear in the main function list. The EMAINFUN tag is required and closes each specification of MAINFUN.

:MAINFUN tag attributes

Syntax	**Attributes**		
:program : :	*:MAINFUN*		
:mainfun : :	**Attributes** NAME=	**Values** *name*	**Uses** Identifies a function part
:emainfun	[.[*flow statements*]]		Specifies the flow statements associated with a function

Syntax	Attributes		
: :			
:eprogram	***:EMAINFUN*** [.]		

NAME=

name Identifies a function part

NAME consists of 1 to 18 characters and must meet the following conventions:

- The first character is alphabetic or national (A-Z, $, #, @)
- The remaining characters are alphanumeric or national characters, hyphens, or underscores (A-Z, 0-9, $, #, @, -, _)
- The name cannot contain blanks or have an EZE prefix.

The name can be a DBCS name up to 8 DBCS characters long with no embedded blanks.

flow statements

flow statements
Specifies the flow statements, if any, associated with a function

Flow statements can consist of multiple lines of text. A statement line cannot exceed 2000 bytes.

Table and additional records list definition

The TABREC tag, which can be repeated, specifies the following:

- Each table used in the program, including edit tables for data items on a map. The list of tables is used to verify references to tables by processing statements and to assure the tables are available when the program runs. This list will not include message tables picked up because of the message table prefix specified with the program.
- Each record used as an additional working storage area, an argument passed on a call, or a record redefinition referred to in the program that is not the object of one of the functions in the program.

This optional tag with its attributes further describes the PROGRAM and EPROGRAM tag set.

:TABREC tag attributes

Syntax	Attributes		
:program : :	*:TABREC*		
:tabrec : :	**Attributes** NAME=	**Values** *name*	**Uses** Specifies the name of the table or additional record
:eprogram	TYPE= [.]	{RECORD | TABLE}	Specifies the type of part

NAME=

name Specifies the name of the table or additional record

NAME depends on the TYPE specification. If the NAME is the name of a record part, the name consists of 1 to 18 characters and must meet the following conventions:

- The first character is alphabetic or national (A-Z, $, #, @)
- The remaining characters are alphanumeric or national characters, hyphens, or underscores (A-Z, 0-9, $, #, @, -, _)
- The record name cannot contain blanks or have an EZE prefix.

The record name can be a DBCS name up to 8 DBCS characters long with no embedded blanks.

If the NAME is the name of the table part, it consists of 1 to 7 characters and must meet the following conventions:

- The first character is alphabetic (A-Z)
- The remaining characters are alphanumeric (A-Z, 0-9)
- The table name cannot contain blanks or have an EZE prefix
- Table names cannot end in 0.

TYPE=

RECORD
Specifies that the additional part is a record

TABLE
Specifies that the part is a table

Parameter specification

The CALLPARM tag specifies the name and type of each parameter received by a called program. This optional tag with its attributes further describes the PROGRAM and EPROGRAM tag set.

The maximum number of parameters is 30. Parameters can be maps, records, or data items. List the parameters in the same order as the arguments are listed in the program's CALL statement. The number of parameters must equal the number of arguments.

The parameter definitions are the same as, or compatible with, the definitions of the call arguments. If data types are not compatible or lengths are not the same, errors might occur during execution. A data item parameter cannot be a data item in a record, table, or map used by the called program. You can define the data item using data item definition.

:CALLPARM tag attributes

Syntax	**Attributes**		
:program . .	***:CALLPARM***		
:callparm . .	**Attributes** NAME=	**Values** *name*	**Uses** Specifies the name of the parameter
:eprogram	[TYPE= [.]	{RECORD|ITEM|MAP}]	Specifies the type of parameter

NAME=

name Specifies the name of the parameter

NAME depends on the TYPE specification. If NAME is the name of a record part, the name consists of 1 to 18 characters and must meet the following conventions:

- The first character is alphabetic or national (A-Z, $, #, @)
- The remaining characters are alphanumeric or national characters, hyphens, or underscores (A-Z, 0-9, $, #, @, -, _)
- The record name cannot contain blanks or have an EZE prefix.

The record name can be a DBCS name up to 8 DBCS characters long with no embedded blanks.

If NAME is the name of a data item part, it consists of 1 to 32 characters and must meet the following conventions:

- The first character is alphabetic or national (A-Z, $, #, @)
- The remaining characters are alphanumeric or national characters, hyphens, or underscores (A-Z, 0-9, $, #, @, -, _)
- The item name cannot contain blanks or have an EZE prefix.

The item name can be a DBCS name up to 15 DBCS characters long with no imbedded blanks.

Note: EZEDLPSB and EZEDLPCB can be specified as item names but cannot both be specified in the same called parameter list. EZEDLPCB can be subscripted with a numeric literal.

If NAME is the name of a map part, it consists of 1 to 8 characters and must meet the following conventions:

- The first character is alphabetic or national (A-Z, $, #, @)
- The remaining characters are alphanumeric or national characters (A-Z, 0-9, $, #, @)
- The map name cannot contain blanks or have an EZE prefix.

TYPE=

RECORD
Specifies that the parameter is a record

ITEM Specifies that the parameter is a data item

MAP Specifies that the parameter is a map

Program prologue definition

The PROL tag is a text description of a defined program. This optional tag further describes the PROGRAM and EPROGRAM tag set. The EPROL tag closes the description, and if you use the PROL tag, the EPROL tag is required.

:PROL tag values

Syntax	Usage
:program	

Syntax	Usage	
. .	*:PROL*	
:prol . .	**Value** [.[*prologue lines*]]	**Usage** Describes the program
:eprol . .	*:EPROL* [.]	
:eprogram		

Only the library size and machine size limit the number of lines that you can specify. Each line can contain up to 60 characters. You can use both uppercase and lowercase in the prologue text. The text is saved as entered. When the text is longer than 60 characters, it is split into multiple lines during import.

Templates tracability information

Templates tracability information is defined for a program part as described in "Chapter 16. Templates traceability information structures" on page 213.

Program definition syntax example

```
:program    name     = xxxxxxx
           date      = 'mm/dd/yyyy'  time = 'hh:mm:ss'  type = xxxxxxxxx
           execmode  = xxxxxxxxxxxx
           workstor  = xxxxxxxxxxxxxxxxxx   mapgroup = xxxxxx
           helpgrp   = xxxxxx               helpkey  = xx
           psb       = xxxxxxxx             pfequate = x
           implicit  = x                    msgtable = xxxx
           bypkey    = xx xx xx xx xx       firstmap = xxxxxxxx
           firstui   = xxxxxxxxxxxxxxxxxx.
:mainfun   name     = xxxxxxxxxxxxxxxxxx.
MOVE A TO B;
:emainfun.
:tabrec    name     = xxxxxxxxxxxxxxxxx
           type     = xxxxxx.
:callparm  name     = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
           type     = xxxx.
:prol.
Each prologue line can be up to 60 characters long.  IMPORT
will split lines longer than 60 characters.
:eprol.
:eprogram.
```

Note: This example illustrates the syntax of all program definition tags and attributes. Actual definitions do not require or use all tags and attributes shown here.

Chapter 3. GUI client structures

This chapter describes the tags and attributes that define a VisualAge Generator GUI client.

Note: GUI clients are only valid on import.

When you import GUI client parts in character format, be aware that the character format uses some characters that might not be in the same place on all code pages. You must either use this function on machines with the same code page or on machines whose code pages hold these characters in the same ASCII position.

GUI client definition

The GUIAPP tag defines attributes for individual GUI client parts. There are two ways to represent the structure of a GUI client program:

GUI interchange format
Supports the exchange of the GUI client definition with other tools. The SCRIPT and ESCRIPT tags are used to specify the GUI interchange format.

Unloaded format
Provides fast VisualAge Generator import, but it cannot be exchanged with other tools. The BENCODE and EBENCODE tags are used to specify the unloaded format.

Either the SCRIPT tag or the BENCODE tag is required. The EGUIAPP tag closes the definition and is required.

:GUIAPP tag attributes

Syntax	**Attributes**		
:guiapp ⋮	***:GUIAPP***		
:script	**Attributes**	**Values**	**Uses**
⋮	[DATE=	*'modification date'*]	Specifies the date the GUI client was last modified
:escript	NAME=	*GUI client name*	Specifies the name of the GUI client

Syntax	Attributes		
: :	[TIME=	*'modification time'*]	Specifies the time the GUI client was last modified
or	[.]		
: :	***:EGUIAPP***		
:bencode : :			
:ebencode : :			
:eguiapp			

DATE=

modification date
Specifies the date the GUI client part was last modified

DATE format is mm/dd/yyyy. Single quotation marks are optional.

NAME=

GUI program name
Identifies the name of the GUI client

TIME=

modification time
Specifies the time the GUI client part was last modified

TIME format is hh:mm:ss. Single quotation marks are optional.

GUI interchange format

The SCRIPT tag begins the set of GUI interchange format statements that make up the GUI client definition. The ESCRIPT tag closes the statement and is required.

The GUI client definition statements are in the IBM Smalltalk language syntax. The statements are represented using the IBM Smalltalk external source format.

Refer to the *IBM Smalltalk Programmer's Reference*, SC34-4493, and the *Introduction to Object-Oriented Programming with IBM Smalltalk* document, SC34-4491, for information about IBM Smalltalk language.

:SCRIPT tag attributes

Syntax	Usage	
:guiapp . . .	***:SCRIPT***	
:script . . .	**Values** [.[*GUI interchange format statements*]]	**Uses** Specifies statements that describe the GUI definition
:escript . . .	***:ESCRIPT*** [.]	
:eguiapp		

Statements can contain character data and white space. Character data is case sensitive. White space is represented by any of the following codes: space, carriage return, tab, line feed, and form feed. You cannot use the null character. A statement can span multiple lines of text and a line can be any length.

Three categories of information are provided in a GUI client definition: the public interface, the internal structure, and the visual layout. This information is stored in the GUI client part in an object-oriented way as objects.

The GUI interchange format contains statements to recreate the objects that represent the GUI client part. These statements are defined as class methods of the GUI client.

The following table shows the objects, the class methods, and indicates whether the methods are optional or required for a GUI client definition.

Object	Method	Required/ Optional
Interface specification Defines the features that are added to the public interface of the GUI client	**calculateInterfaceSpec** Recreates the interface specification for the GUI client	Optional
Part builder Defines the internal structure of the GUI client, which consists of its subpart tree, values for attribute settings, and connections between subparts	**calculatePartBuilder** Recreates the part builder for the GUI client	Required
Visual layout part builder Defines the placement of non-visual components on the free-form surface	**calculateVisualLayoutPartBuilder** Recreates the visual layout part builder for the GUI client	Optional
Note: The calculateVisualLayoutPartBuilder method is not normally used to interchange with other tools. If omitted, GUI client definition will place parts and connections on the free-form surface by default.		

The methods appear in the following sequence:
- publicMethods
  - calculateInterfaceSpec
  - calculatePartBuilder
  - calculateVisualLayoutPartBuilder
- PrivateMethods

Private methods are only included if they are part of the implementation of the public methods.

To determine the statements that represent a specific aspect of a GUI client, use VisualAge Generator Developer to define a sample program, then export the GUI client in GUI interchange format and browse or print the output file.

GUI unloaded format

The BENCODE tag begins the binary code that represents the GUI client definition. The EBENCODE tag closes the statement and is required.

:BENCODE tag attributes

Syntax	Usage	
:guiapp		

Syntax	Usage	
: :	*:BENCODE*	
:bencode : :	**Values** [.[*GUI unloaded format*]]	**Uses** Specifies the GUI client definition in a binary format
:ebencode : :	*:EBENCODE* [.]	
:eguiapp		

GUI client definition syntax example

```
:guiapp    name      = xxxxxxxx
           date      = 'mm/dd/yyyy'      time = 'hh:mm:ss'
:script.
    GUI definition in Interchange Format
:escript.
:eguiapp.
```

Note: This example illustrates the syntax of some of the GUI client definition tags and attributes. Actual definitions do not require or use all tags and attributes shown here.

Chapter 4. Function structures

This chapter describes the external source format tags and attributes that define a VisualAge Generator function.

Function definition

The FUNC tag replaces both the PROCESS and GROUP tags, which are logic parts used in previous versions of VisualAge Generator. The PARM, STORAGE, RETURN, BEFORE, AFTER, SQL, DLICALL, SSA, and QUAL tags further define aspects of a function and can have attributes. The EFUNC tag closes the definition, and if you use the FUNC tag, the EFUNC tag is required.

:FUNC tag attributes

Syntax	**Attributes**		
:func ⋮ ***:FUNC***			
:parm ⋮			
	Attributes	**Values**	**Uses**
:storage ⋮	[DATE=	'*modification date*']	Specifies the date the function part was last modified
:return ⋮	[DESC= [ERRRTN=	'*description*'] *routine name*]	Describes the function Specifies an error-handling routine
:before ⋮ :ebefore	[EXECBLD=	{Y | **N**}]	Specifies whether the SQL statements are to be built as static SQL statements or prepared as dynamic statements
:after ⋮	[MODEL= NAME=	{DELETE | UPDATE | **NONE**}] *name*	Specifies generation of a model SQL statement Identifies the function part
:eafter	[OBJECT=	*object name*]	Specifies a record or map accessed by OPTION

Syntax	Attributes		
:sql : :	[OPTION=	{**EXECUTE** | CONVERSE | DISPLAY | INQUIRY | UPDATE | REPLACE | ADD | DELETE | SCAN | CLOSE | SETINQ | SETUPD | SQLEXEC | SCANBACK}]	Specifies the input/output logic in a function
:esql	[REFINE=	{Y | **N**}]	Specifies the method of displaying logic for the function
:dlicall	[SINGROW=	{Y | N}]	Specifies whether a single-row SELECT is to be performed for an SQL INQUIRY
: : :	[TIME=	*'modification time'*]	Specifies the time the function part was last modified
:ssa	[UPDFUNC=	*update function name*]	Specifies the name of an UPDATE or SETUPD function
: : :	[WITHHOLD=	{Y | N}]	Specifies whether the DECLARE CURSOR statement issued for a SETINQ or SETUPD I/O option includes a WITH HOLD clause
:qual			
: : :	[.]		
:vagt : :	***:EFUNC*** [.]		
:tracbag : :			
:etracbag **:efunc**			

DATE=

'modification date'
Specifies the date the function part was last modified

DATE format is mm/dd/yyyy. Single quotation marks are optional.

DESC=

'description'
Describes the function

DESC consists of 1 to 30 characters that can be entered in uppercase and lowercase. Mixed data is also permitted. A function description is optional; it does not affect execution.

ERRRTN=

routine name
Specifies an error-handling routine

ERRRTN calls an error routine when an error occurs during execution of a function option that accesses a record. ERRRTN can be a main function or the special function words EZERTN, EZEFLO, or EZECLOS. For information on special function words, see the VisualAge Generator Developer help system.

If the error routine is a main function, control is transferred to that function if an error occurs. Otherwise, control returns to the statement following the I/O option after the error routine returns. If no error routine is specified, a program ends when an error occurs and a message appears describing the error condition.

Error routines cannot be specified for functions with map I/O objects or for EXECUTE functions. DISPLAY or CONVERSE errors cause the program to end.

EXECBLD=

Y Prepares and executes SQL statements as dynamic or extended dynamic statements

N Uses the execution mode specified at generation or execution (default value)

EXECBLD can only be used with the following I/O options:
- INQUIRY
- SETINQ

- SETUPD
- SQLEXEC
- UPDATE

MODEL=

DELETE
Specifies that a default DELETE SQL statement is to be generated

UPDATE
Specifies that a default UPDATE SQL statement is to be generated

NONE
Specifies that no model statement is to be generated (default value)

You can specify MODEL for SQLEXEC functions only.

NAME=

name Identifies a function part

NAME consists of 1 to 18 characters and must meet the following conventions:

- The first character must be alphabetic or national (A-Z, $, #, @)
- The remaining characters must be alphanumeric or national characters, hyphens, or underscores (A-Z, 0-9, $, #, @, -, _)
- The name cannot contain blanks or have an EZE prefix.

The name can be a DBCS name up to 8 DBCS characters long with no embedded blanks.

OBJECT=

object name
Specifies a record or map accessed by OPTION

EXECUTE functions do not have an object, and SQLEXEC functions may or may not have an object. Other functions must have an object.

OPTION=

option Specifies the input/output logic in a function

OPTION can be specified as any one of the following I/O options:

ADD places a new record in a file or database. The program should initialize all fields in the record prior to executing the ADD option.

CLOSE
closes a file, disconnects a printer, or releases any unprocessed rows in a set of SQL row records selected by UPDATE, SETUPD, or SETINQ.

CONVERSE
displays a map at a terminal and waits for input to be entered by the user at the terminal.

DELETE
removes a record from a file or database.

DISPLAY
sends a map to a printer or to a terminal output buffer.

EXECUTE
is not associated with an input/output operation. It can be used for special processing such as control of flow between functions, initialization, processing not to be repeated in an I/O function, error handling, and termination processing. EXECUTE is the default OPTION value.

INQUIRY
reads a single record from a file or database.

REPLACE
puts a changed record back into a file or database.

SCAN reads the next record in a file or database.

SCANBACK
reads the previous record in an indexed file.

SETINQ
selects a set of rows from a relational database for later retrieval with the SCAN I/O option. The object must be an SQL row record.

SETUPD
selects a set of rows from a relational database for faster processing with the SCAN I/O option. The records selected can be replaced or deleted. The object must be an SQL row record.

SQLEXEC
executes an SQL statement that you define by using the SQL tag.

UPDATE
reads a record from a file or database with the implied intention of replacing or deleting the record.

REFINE=

Y	Displays the subtree of a function part in the structure diagram
N	Prevents the display of the subtree of a function part in the structure diagram (default value)

REFINE corresponds to the default collapse attribute in VisualAge Generator Developer. It prevents the details logically beneath the function part from showing in the structure diagram. Default collapse for a function part is only in effect when you select the default collapse attribute for the program structure diagram. The attribute is preserved in the library, but it has no effect when the generated program is run.

SINGROW=

Y	Specifies that a single-row SELECT is to be performed for an SQL INQUIRY
N	Specifies that a single-row SELECT is not to be performed

SINGROW can be used only when a program is running in static mode (on DB2 systems). The single row SELECT is not available when running VisualAge Generator SQL programs in dynamic mode. SINGROW is not valid when an ORDERBY clause exists. The default is Y.

TIME=

'modification time'
Specifies the time that the function part was last modified

TIME format is hh:mm:ss. Single quotation marks are optional.

UPDFUNC=

function name
Specifies the name of an UPDATE or SETUPD function that selected the rows to be replaced by this function

UPDFUNC can be specified for a REPLACE function.

WITHHOLD=

Y Specifies that the cursor is not closed as a consequence of a commit function. A SCAN option issued after the commit retrieves the next row in the selected set of rows.

N Specifies that the cursor is closed after a commit.

If WITHHOLD is used with the SETUPD function option, a SCAN must be issued after a commit before a REPLACE or DELETE option can be issued for the row. WITHHOLD applies only to the SETINQ and SETUPD function options.

WITHHOLD is not effective on rollback and connect functions at the end of a segment when running on CICS or IMS. WITHHOLD is not supported on DB2/VSE.

The default is Y.

Parameter definition

The PARM tag specifies the name and type of each parameter passed to a function. Any number of parameters may be specified. This optional tag further defines the FUNC and EFUNC tag set.

:PARM tag attributes

Syntax	**Attributes**		
:func ⋮ ⋮	***:PARM***		
	Attributes	**Values**	**Uses**
:parm	[BYTES =	*length in bytes*]	Specifies the field length in bytes based on the data item type

Syntax	Attributes		
: :	[DECIMALS =	{{**0** | *decimal places*}]	Specifies the number of decimal places
	[DESC =	*'description'*]	Specifies the description of the parameter
	NAME	*name*	Specifies the name of the parameter
	[PARMTYPE	{RECORD | **ITEM** | MAPITEM | SQLITEM}]	Specifies the type of the parameter and how it it to be used
	[TYPE =	{ANYCHA | ANYDBCS | ANYHEX | ANYMIX | ANYNUMERIC | ANYUNICODE | BIN | **CHA** | DBCS | HEX | MIX | NUM | NUMC | PACK | PACF | UNICODE}]	Specifies the data type for the parameter
	[USAGE =	{SHARED | **NONSHARED**}]	Specifies the location of the definition
:efunc			

BYTES=

length in bytes
Specifies the length in bytes based on the data item type.

3 Default for CHA, MIX, NUM, and NUMC

2 Default for BIN, PACK, PACF, and HEX

6 Default for DBCS, UNICODE

BYTES is the number of bytes required to store a data item value internally. The following table lists the valid BYTES values by data type.

Item Type	Byte Values
CHA, HEX, MIX	1-32,767
DBCS, UNICODE	2-32,766 (even only)
NUM, NUMC	1-18
PACK, PACF	1-10
BIN	2, 4, 8 only

Note: The BYTES attribute can be specified only when PARMTYPE=ITEM, MAPITEM, or SQLITEM, USAGE=NONSHARED, and TYPE is not one of the ANY types.

DECIMALS=

0 Specifies no decimal places (default value)

decimal places
Specifies the number of decimal places

DECIMALS is the number of positions to the right of an implied decimal point in numeric items. The length specified for the data item must include the places set aside for decimal position. The maximum number of decimal positions is 18 or the number of digits defined for the data item, whichever is smaller. The decimal point is not stored with the data.

Note: The DECIMALS attribute can be specified only when PARMTYPE=ITEM, MAPITEM, or SQLITEM, USAGE=NONSHARED, and TYPE is not CHA, DBCS, MIX, HEX, UNICODE or any one of the ANY types.

DESC=

'description'
DESC describes what the parameter represents.

DESC is a text description of what the parameter represents. The text can consist of up to 30 characters, not including the quotation marks, and can be entered in uppercase and lowercase. Mixed data is also permitted.

NAME =

name Specifies the name of the parameter

EZEwords cannot be specified as parameters.

PARMTYPE=

RECORD
Specifies that the parameter is a record. The library part must be a working storage record. The corresponding argument must be a record with length at least as long as the parameter record.

ITEM Specifies that the parameter is an item. The corresponding argument must be an item with a compatible definition or a literal. (default value)

MAPITEM
Specifies that the parameter is a map item and has map item state information. The corresponding argument must be a map item with a compatible definition.

SQLITEM
Specifies that the parameter is an SQL item and has SQL state information. The corresponding argument must be an SQL item with a compatible definition.

PARMTYPE specifies the type of parameter that is expected to be passed to the function. MAPITEM and SQLITEM indicate that certain state information is available to be tested for or set in the logic of the function.

TYPE=

ANYCHAR
Character data. The corresponding argument must have type CHA with any length.

ANYDBCS
DBCS data. The corresponding argument must have type DBCS with any length.

ANYHEX
Hexadecimal data. The corresponding argument must have type HEX with any length.

ANYMIXED
Mixed data. The corresponding argument must have type MIX with any length.

ANYNUMERIC
Numeric item. The corresponding argument must have a numeric type (BIN, NUM, NUMC, PACK, PACF). Any length or number of decimal places is allowed.

ANYUNICODE
Unicode character data (only valid for Developer on JAVA). The corresponding argument must have type UNICODE with any length.

BIN Binary number

CHA Character data (default value)

DBCS Double-byte character data

HEX Hexadecimal data

MIX DBCS data intermingled with single-byte character data

NUM Numeric characters with positive sign in F format

NUMC
Numeric characters with positive sign in C format

PACF Packed decimal characters with positive sign in F format

PACK Packed decimal characters with positive sign in C format

UNICODE
Unicode character data (only valid for Developer on JAVA)

TYPE specifies the internal format or type of data. The data type determines how the item is processed when referred to in processing statements.

Defining item parameters as one of the ANY item types specifies loose typing of the parameter. Bytes, decimals, and a specific numeric type are not specified for these parameters. Instead, test and generation will determine the appropriate conversion that needs to occur between the argument and the parameter. This may result in multiple copies of the function being produced in the generated code.

Defining item parameters with a data type other than one of the ANY item types specifies strong typing of the parameter. Bytes, decimals, and a specific numeric type are either specified or defaulted for you. Test and generation will require exact matches between arguments and parameters when strong typing is used.

For map item parameters, the only valid parameter item types are: NUM, CHA, DBCS, MIX, ANYNUMERIC, ANYCHAR, ANYDBCS, and ANYMIX.

Note: The TYPE attribute can be specified only for PARMTYPE=ITEM, MAPITEM, or SQLITEM, and USAGE=NONSHARED.

USAGE=

SHARED
Indicates that the definition of the item is shared with other references. The item definition stored in the library is used.

NONSHARED
Indicates that the definition of the item is unique to this use. The item definition stored in the library is not used. (default value)

Note: The USAGE attribute can be specified only when PARMTYPE=ITEM, MAPITEM, or SQLITEM.

Local Storage Definition

The STORAGE tag specifies the name and type of each local storage area defined for the function. Any number of local storage areas may be specified. This optional tag further defines the FUNC and EFUNC tag set.

:STORAGE tag attributes

Syntax	Attributes		
:func			

Syntax	Attributes		
⋮	*:STORAGE*		
	Attributes	**Values**	**Uses**
:storage	[BYTES=	*length in bytes*]	Specifies the length in bytes for the data item based on the data item type
	[DECIMALS=	{**0** | *decimal places*}]	Specifies the number of decimal places
⋮	[DESC=	*'description'*]	Specifies the description of the local storage area
	NAME=	*name*	Specifies the name of the local storage area
	[STORTYPE =	{RECORD | **ITEM**}]	Specifies the type of the storage area
	[TYPE=	{BIN | **CHA** | DBCS | HEX | MIX | NUM | NUMC | PACK | PACF | UNICODE}]	Specifies the data type for a local storage item
	[USAGE=	{SHARED | **NONSHARED**}]	Specifies the location of the item definition
:efunc			

BYTES =

length in bytes
Specifies the length in bytes for the data item based on the data item type.

3 Default for CHA, MIX, NUM, and NUMC

2 Default for BIN, PACK, PACF, and HEX

6 Default for DBCS, UNICODE

BYTES is the number of bytes required to store a data item value internally. The following table lists the valid BYTES values by data type.

Item Type	Byte Values
CHA, HEX, MIX	1-32,767
DBCS, UNICODE	2-32,766 (even only)
NUM, NUMC	1-18
PACK, PACF	1-10

Item Type	Byte Values
BIN	2, 4, 8 only

Note: The BYTES attribute can be specified only for STORTYPE=ITEM, USAGE=NONSHARED.

DECIMALS=

0 Specifies no decimal places (default value)

decimal places
Specifies the number of decimal places

DECIMALS is the number of positions to the right of an implied decimal point in numeric items. The length specified for the data item must include the places set aside for decimal position. The maximum number of decimal positions is 18 or the number of digits defined for the data item, whichever is smaller. The decimal point is not stored with the data.

Note: The DECIMALS attribute can be specified only when STORTYPE=ITEM, USAGE=NONSHARED, and TYPE is not CHA, DBCS, MIX, HEX or UNICODE.

DESC=

'description'
DESC describes what the local storage area represents.

DESC is a text description of what the local storage area represents. The text can consist of up to 30 characters, not including the quotation marks, and can be entered in uppercase and lowercase. Mixed data is also permitted.

NAME=

name Specifies the name of the local storage

EZEwords cannot be specified as local storage areas.

STORTYPE=

RECORD
Specifies that the local storage is a working storage record

ITEM Specifies that the local storage is an item. (default)

STORTYPE specifies the type of storage area to be defined for use by the function. The area defined is known only to the function in which it is named.

If the storage type is RECORD, the name must be the name of a working storage record part in the library. If the storage type is ITEM, this names a data item which can only be referenced within this function.

TYPE=

BIN Binary number

CHA Character data (default)

DBCS Double-byte character data

HEX Hexadecimal data

MIX DBCS data intermingled with single-byte character data

NUM Numeric characters with positive sign in F format

NUMC
Numeric characters with positive sign in C format

PACF Packed decimal characters with positive sign in F format

PACK Packed decimal characters with positive sign in C format

UNICODE
Unicode character data (only valid for Developer on JAVA)

TYPE specifies the internal format or type of data. The data type determines how the item is processed when referred to in processing statements.

Note: The TYPE attribute can be specified only for STORTYPE=ITEM, USAGE=NONSHARED.

USAGE=

SHARED
Indicates that the definition of the item is shared with other references. The definition stored in the library is used.

NONSHARED
Indicates that the definition of the item is unique to this use. The definition stored in the library is not used. (default value)

Note: The USAGE attribute can be specified only when STORTYPE=ITEM.

Return Value Definition

The RETURN tag specifies the characteristics of the return value of the function. This tag further defines the FUNC and EFUNC tag set. The RETURN tag can appear only once within a function definition.

:RETURN Tag Attributes

Syntax	**Attributes**		
:func : :	***:RETURN***		
	Attributes	**Values**	**Uses**
:return : :	[BYTES=	*length in bytes*]	Specifies the length in bytes for the return value based on the data item type
	[DECIMALS=	{**0**|*decimal places*}]	Specifies the number of decimal places
	[DESC=	'description']	Specifies the description of the return value
	[TYPE=	{BIN |**CHA** |DBCS |HEX |MIX |NUM |NUMC |PACK |PACF |UNICODE}]	Specifies the data type for the return value
:efunc			

BYTES=

length in bytes
Specifies the length in bytes for the return value based on the data item type

3 Default for CHA, MIX, NUM, and NUMC

2 Default for BIN, PACK, PACF, and HEX

1 Default for DBCS, UNICODE

BYTES is the number of bytes required to store a data item value internally. The following table lists the valid BYTES values by data type.

Item Type	Byte Values
CHA, HEX, MIX	1-32,767
DBCS, UNICODE	2-32,766 (even only)
NUM, NUMC	1-18
PACK, PACF	1-10
BIN	2, 4, 8 only

DECIMALS=

0 Specifies no decimal places (default value)

decimal places
Specifies the number of decimal places

DECIMALS is the number of positions to the right of an implied decimal point in numeric items. The length specified for the data item must include the places set aside for decimal positions. The maximum number of decimal positions is 18 or the number of digits defined for the data item, whichever is smaller. The decimal point is not stored with the data.

Note: The DECIMALS attribute can be specified only when the TYPE is not CHA, DBCS, MIX, HEX, or UNICODE.

DESC=

DESC DESC describes what the return value represents.

DESC is a text description of what the data item represents. The text can consist of up to 30 characters, not including the quotation marks, and can be entered in uppercase and lowercase. Mixed data is also permitted.

TYPE=

BIN Binary number

CHA Character data (default value)

DBCS Double-byte character data

HEX Hexadecimal data

MIX DBCS data intermingled with single-byte character data

NUM Numeric characters with positive sign in F format

NUMC
Numeric characters with positive sign in C format

PACF Packed decimal characters with positive sign in F format

PACK Packed decimal characters with positive sign in C format

UNICODE
Unicode character data (only valid for Developer on JAVA)

TYPE specifies the internal format or type of data. The data type determines how the item is processed when referred to in processing statements. Return values are defined with strong typing. Bytes, decimals, and a specific numeric type are either specified or defaulted for you. Upon exit from the function, the return value is assigned to the receiving data area according to move compatibility rules.

Logic definition before I/O option

The BEFORE tag begins the set of statements that perform logic before the I/O option is executed. This optional tag further defines the FUNC and EFUNC tag set. The EBEFORE tag closes the definition, and if you use the BEFORE tag, the EBEFORE tag is required.

The BEFORE tag can appear only once within a function definition. The statement syntax is VisualAge Generator statement syntax. Validation is for statement syntax only, not for part usage. Statements are limited to one per line; however, a single statement can span multiple lines.

```
:
:before. before processing statement1
         before processing statement2
   :
         before processing statementn
:ebefore.
:
```

For an EXECUTE function, all statements specified on the BEFORE tag are executed before any statements on the AFTER tag.

:BEFORE tag values

Syntax	Usage	
:func : :	***:BEFORE***	
:before : :	**Values** [.[*before processing statements*]]	**Uses** Specifies statements that perform logic before the I/O option is executed
:ebefore : :	***:EBEFORE*** [.]	
:efunc		

Statements can consist of multiple lines of text. A statement line cannot exceed 2000 bytes.

Logic definition after I/O option

The AFTER tag begins the set of statements that perform logic after the I/O option is executed. This optional tag further defines the FUNC and EFUNC tag set. The EAFTER tag closes the definition, and if you use the AFTER tag, the EAFTER tag is required.

The AFTER tag can appear only once within a function definition. The statement syntax is VisualAge Generator statement syntax. Validation is for statement syntax only, not for part usage. Statements are limited to one per line; however, a single statement can span multiple lines.

```
:
```

```
:after. after processing statement1
        after processing statement2
   .
   .
   .
        after processing statementn
:eafter.
.
.
.
```

For an EXECUTE function, all statements specified on the AFTER tag are executed after any statements on the BEFORE tag.

:AFTER tag values

Syntax	Usage	
:func . . .	*:AFTER*	
:after . . .	**Values** [.[*after processing statements*]]	**Uses** Specifies statements that perform logic after the I/O option is executed
:eafter . . .	*:EAFTER* [.]	
:efunc		

Statements can consist of multiple lines of text. A statement line cannot exceed 2000 bytes.

SQL selection condition specification

The SQL tag specifies the SQL selection conditions that VisualAge Generator uses to retrieve information from the SQL database. This optional tag further defines the FUNC and EFUNC tag set. The ESQL tag closes the specification, and if you use the SQL tag, the ESQL tag is required.

The SQL tag is valid for the INQUIRY, SETINQ, SETUPD, SQLEXEC, UPDATE, ADD, and REPLACE I/O options. The SQL tag is not valid for the EXECUTE, CONVERSE, DISPLAY, and SCANBACK I/O options because they cannot have an SQL row record as the I/O object.

You can repeat the SQL tag for each SQL clause that you specify, but the tag can appear only once for each type of clause.

:SQL tag attributes

Syntax	Attributes		
:func ⋮	*:SQL*		
:sql ⋮	**Attributes** CLAUSE=	**Values** {SELECT | INTO | SET | WHERE | ORDERBY | SQLEXEC | FORUPDATEOF | VALUES | INSERTCOLNAME}	**Uses** Specifies the type of clause described by this tag
:esql	[HOSTVAR=	{'?' | '@'}]	Specifies the character used within the clause text to indicate the beginning of a host variable name
⋮	[.[*clause text*]]		Specifies the clause for the SQL statement
:efunc	*:ESQL* [.]		

CLAUSE=

SELECT
Specifies the SELECT clause

INTO Specifies the INTO clause

SET Specifies the SET clause

WHERE
Specifies the WHERE clause

ORDERBY
Specifies the ORDER BY clause

SQLEXEC
Specifies the SQLEXEC clause

VALUES
Specifies the VALUES clause

FORUPDATEOF
Specifies the FOR UPDATE OF clause

INSERTCOLNAME
Specifies the column names for the INSERT INTO clause

Certain CLAUSE specifications are valid for certain I/O options. The following table identifies the applicable specifications:

I/O option
CLAUSE specification

INQUIRY
SELECT, INTO, WHERE, ORDERBY

SETINQ
SELECT, INTO, WHERE, ORDERBY

UPDATE
SELECT, INTO, WHERE, FORUPDATEOF

SETUPD
SELECT, INTO, WHERE, FORUPDATEOF

REPLACE
SET

SQLEXEC
SQLEXEC

ADD VALUES, INSERTCOLNAME

HOSTVAR=

'?' Specifies that a ? is used to begin host variable names

'@' Specifies that an @ is used to begin host variable names (default value)

Export always uses a ? as the host variable name indicator so that an @ can be properly interpreted as a valid character in a name. When variable names contain an @ and the @ is also used as the host variable name indicator, unpredictable results can occur on import.

clause text

clause text
Specifies the clause for the SQL statement

The clause text for the SELECT, INTO, SET, VALUES, INSERTCOLNAME, and FORUPDATEOF clauses do not include their respective keywords.

The clause text for a WHERE clause is the clause that includes the WHERE keyword. The WHERE clause can contain the ORDER BY clause if the ORDER

BY clause was not placed on a separate line during program definition of the SQL statement. A WHERE clause without any clause text indicates that there is no WHERE clause in the SQL statement.

The clause text for an ORDER BY clause is the clause that includes the ORDER BY keywords. An ORDER BY clause without any clause text indicates that there is no ORDER BY clause in the SQL statement.

The clause text for an SQLEXEC clause is the entire clause including all keywords.

Within the clause text, SQL column names always begin with an exclamation point (!). When produced from VisualAge Generator export, host variable names always begin with a question mark (?).

Statements can consist of multiple lines of text. A statement line cannot exceed 2000 bytes.

DL/I call definition

The DLICALL tag begins the definition of calls to access data stored in DL/I databases. This optional tag with its attributes further describes the FUNC and EFUNC tag set.

:DLICALL tag attributes

Syntax	**Attributes**		
:func ⋮	***:DLICALL***		
:dlicall ⋮	**Attributes**	**Values**	**Uses**
	DBDNAME=	{*database name* | *number*}	Specifies the database in the application program specification block that is to be accessed by this DL/I call
:efunc	[MODIFIED=	{Y | N}]	Specifies whether the segment search argument (SSA) was modified
	PSB=	*PSB name*	Specifies a program specification block used by this function
	[SCANPAR=	{Y | **N**}]	Specifies whether the range of the SCAN is limited to the currently established parent chain

Syntax	Attributes		
	[SCANUPD=	{Y | **N**}]	Specifies whether a segment retrieved by a SCAN function can be replaced or deleted
	[.]		

DBDNAME=

database name
Accesses the first program communication block in the program specification block with this database name

number
Accesses this number program communication block in the program specification block

DBDNAME identifies the database in the program specification block that is to be accessed by this DL/I call.

MODIFIED=

Y Specifies that the segment search argument (SSA) was modified

N Specifies that the segment search argument (SSA) was not modified

If the Segment Search Argument (SSA) tag is specified, the default is Y. If the SSA tag is not specified, the default is N.

PSB=

PSB name
Specifies a program specification block used by this function

SCANPAR=

Y Specifies that the SCAN range is limited to the current parent chain in the database hierarchy

N Specifies that the next segment of that type in the database is retrieved regardless of parentage (default value)

SCANPAR can be specified only for a DL/I call for a SCAN function.

SCANUPD=

Y Specifies that the program can replace or delete the segments retrieved by the SCAN

N Specifies that the program cannot replace or delete the segments retrieved by the SCAN (default value)

SCANUPD can be specified only for a DL/I call for a SCAN function.

Segment search argument definition

The SSA tag identifies the segments in the database to be accessed on a DL/I call. This optional tag with its attributes further describes the FUNC and EFUNC tag set.

You can specify the SSA tag only if the DLICALL tag is specified. The SSA tag can be repeated.

The order of the SSA tags determines the order of the DL/I segment search arguments for the database call.

:SSA tag attributes

Syntax	**Attributes**		
:func ⋮ ⋮	*:SSA*		
:dlicall ⋮ ⋮	**Attributes** [CMDCODES=	**Values** *'codes'*]	**Uses** Specifies special processing to be performed by the DL/I call

Syntax	Attributes		
:ssa	SEGNAME=	*segment name*	Specifies the DL/I segment accessed by the DL/I call
: :	[.]		
:efunc			

CMDCODES=

'codes'
Specifies special processing to be performed by the DL/I call

You can specify up to four command codes. Refer to the DL/I manuals for a more detailed description of the command codes. The valid command codes are:

C Use the concatenated key to select this segment. When C is specified as a command code, the Segment Field, Boolean Op, and Op fields of the SSA must be left blank. The Comparison Value Item names a data item that contains the entire concatenated key for the segment.

D This code allows the retrieval or insertion of multiple segments in a hierarchical path. This code is not required for the lowest level segment, since it is always retrieved or inserted. Specify this code for any higher level segment to be retrieved on INQUIRY, UPDATE, or SCAN options. For an ADD option, specify this code only for the highest level segment you want inserted, to add that segment and all segments at lower levels.

VisualAge Generator Developer handles I/O buffering for segments retrieved or written using the D command code. If you retrieve multiple segments for update using the D code, a REPLACE option with the lowest level segment as the object will replace all the segments that were retrieved with the D code.

The path call processing option (P) must be specified in DL/I PSB generation if the D command code is used.

F For the SCAN option, start scanning from the first occurrence of this segment type under its parent. For the ADD option, this code is effective only for segments with non-unique or no sequence field, and the segment is inserted at the first position within its parent.

L For INQUIRY, UPDATE, and SCAN options, retrieve the last occurrence of this segment type under its parent. If qualification statements are present, retrieve the last segment that satisfies the

search criteria. For the ADD option, this code is effective only for segments with non-unique or no sequence field, and the segment is inserted at the last position within its parent.

N Do not replace this segment on a replace call even though it was retrieved on the get for update call.

P Set parent position for get next in parent (SCAN) at the hierarchy level represented by this segment.

Q Lock the retrieved segments until checkpoint or PSB termination.

Note: If you used the Q command code in coding DL/I calls for CICS in other languages, you followed the Q command code with an A for IMS compatibility. However, do not enter the A here. VisualAge Generator Developer supplies the A when it builds the final SSA list at execution time.

U Do not move the database position from this segment while searching its hierarchical dependents.

V Like U except that the command code is automatically set at all higher levels in the call.

The following command codes are supported only in the IMS/VS, IMS BMP, and CICS for MVS/ESA environments. Use these codes to access subsets of a special type of database called a fast path data entry database (DEDB). To identify the subset you are accessing, enter the command code followed by an integer from 1 to 8.

M Move subset pointer to next occurrence of the segment in the segment chain.

R Retrieve first occurrence of the segment in the subset.

S Set the subset pointer unconditionally to the current position.

W Set the subset pointer conditionally to the current position.

Z Set the subset pointer to 0.

Certain command codes are applicable only to certain I/O options. The following table identifies the applicable command codes:

Option	Command Codes	Fast Path Command Code
INQUIRY	D, L, Q, U, V, C, P	M, R, S, W, Z
UPDATE	D, L, Q, U, V, C, P	M, R, S, W, Z
ADD	D, L, F, U, V, C	M, R, S, W, Z
REPLACE	N	M, S, W, Z
DELETE	None	Z
SCAN	D, L, F, Q, U, V, C, P	M, R, S, W, Z

Command codes are optional. If none are specified, none are used. The R and F, R and Q, L and F, or U and V command codes cannot both be entered in the command code field for the same SSA. In addition, only one of the M,S,W, and Z command codes can be used in the same SSA.

You can only have one C command code in a set of SSAs. On an INSERT call, the following apply:

- A qualified SSA cannot follow a D command code.
- A C command code cannot follow any SSA with a D command code.

SEGNAME=

segment name
Specifies the DL/I segment accessed by the DL/I call

Specify the name of a segment in the chain that goes from the object segment back to the root segment in the database hierarchy. In addition, define the segment to VisualAge Generator as a DL/I segment record before testing or generating the program.

Qualification statement definition for the segment search argument

The QUAL tag specifies the qualification statement for the segment search argument (SSA) consisting of a segment field, a relational operator, a comparison value, and a Boolean operator. This optional tag with its attributes further describes the FUNC and EFUNC tag set.

You can specify the QUAL tag only if the SSA tag is specified, and you can use it multiple times for each SSA tag.

The order of the QUAL tags determines the order of the qualification statements on the call.

:QUAL tag attributes

Syntax	**Attributes**		
:func : : :dlicall	***:QUAL*** **Attributes**	**Values**	**Uses**

Syntax	Attributes		
:	[BOOLOP=	{& | AND | | | OR}]	Identifies the presence of an additional qualification statement and shows how the true or false values of the qualification statements are to be combined
:ssa	COMPVAL=	*'comparison value'*	Specifies the comparison value item that contains the value to be compared to the contents of the segment field
:	[RELOP=	{=|EQ|>|GT|<|LT|>=|=>|GE|<=|=<|LE|¬=|=¬|NE}]	Specifies the comparison to be done between the segment field and the comparison value item
:qual	[SEGFIELD=	*field name*]	Specifies the field used for segment selection
:	[.]		
:efunc			

BOOLOP=

& or AND
AND operator

\| or OR
OR operator

BOOLOP shows how the true or false values of this qualification statement and the following qualification are to be combined.

If you specified command code C for the SSA, omit this attribute.

COMPVAL=

'comparison value'
Specifies the comparison value item that contains the value to be compared to the contents of the segment field.

At run time, VisualAge Generator uses the value in this item as the field value in building the SSA for the DL/I call. The comparison value is compared to

the contents of the segment field. If the comparison is true, the search criteria of this qualification statement is satisfied.

The maximum length for the comparison value item is 104 characters. If a comparison value item is longer than 71 characters, place a continuation character in column 72. Any character in column 72 is a continuation character. The continuation character causes concatenation of the two lines on import. The two lines concatenate with no blanks between the last character on the first line and the first character on the second line. The length of the concatenated comparison value item can never exceed 104 characters.

COMPVAL always applies when the command code is C.

RELOP=

= or EQ
Equal

> or GT
Greater than

< or LT
Less than

>=, =>, or GE
Greater than or equal

<=, =<, or LE
Less than or equal

¬= , =¬ , or NE
Not equal

Note: NLS Consideration: The "¬" character is not in the National Language syntactic character set; therefore it might not be represented by equivalent code points across different code pages. If you will be transferring your program between machines with differing code pages (particularly between System/370 host systems and workstations), use **NE** rather than "¬=" or "=¬". An alternative for handling codepage-dependent characters is to use the hptcnv30 utility. See *Migration Guide* for more information.

RELOP tells DL/I how to compare the values in the segment field and the comparison value item.

RELOP is required unless you specified command code C for the SSA. If you specified command code C for the SSA, omit the RELOP attribute.

SEGFIELD=

field name
Specifies the field used for segment selection

Specify the field name as defined in the DL/I database description.

SEGFIELD is required unless you specified command code C for the SSA. If you specified command code C for the SSA, omit the SEGFIELD attribute.

Templates tracability information

Templates tracability information is defined for a function part as described in "Chapter 16. Templates traceability information structures" on page 213.

Function definition syntax example

```
:func   name    = xxxxxxxxxxxxxxxxx
           date    = 'mm/dd/yyyy'  time   = 'hh:mm:ss'
           option   = xxxxxxxx           object  = xxxxxxxxxxxxxxxxx
           errrtn   = xxxxxxxxxxxxxxxxx execbld = x   model = xxxxxx
           refine   = x   singrow  = x   updproc = xxxxxxxxxxxxxxxxx
           withhold = x
           desc    = 'This is a function description'.
:parm      name = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           parmtype = XXXXXXX     type = XXXX
           bytes = XXXXX      decimals = XX
           desc    = 'This is a parameter description'
           usage = XXXXXXXXX.
:parm      name = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           parmtype = XXXXXXX     type = XXXX
           bytes = XXXXX      decimals = XX
           desc    = 'This is a parameter description'
           usage = XXXXXXXXX
:storage   name = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           stortype = XXXXXXX     type = XXXX
           bytes = XXXXX      decimals = XX
           desc    = 'A local storage area description'
           usage = XXXXXXXXX.
:return    bytes = XXXXX      decimals = XX
           type = XXXX
           desc    = 'A return value description'.
:before.
MOVE A
TO B;
:ebefore.
:after.
MOVE A
TO B;
:eafter.
:sql       hostvar  ='?'           clause   = xxxxxxxxxxxxx.clause text
:esql.
:dlicall   dbdname  = xxxxxxxx     psb      = xxxxxxxx
           scanupd  = x            scanpar  = x
           modified = x.
:ssa       segname  = xxxxxxxx     cmdcodes = 'xxxx'.
:qual      segfield = xxxxxxxx        relop = xx
           compval  = 'Enter up to 104 characters'
           boolop   = xxx.
:qual      segfield = xxxxxxxx    relop = xx
           compval  = 'Enter up to 104 characters'.
:efunc.
```

Note: This example illustrates the syntax of all function definition tags and attributes. Actual definitions do not require or use all tags and attributes shown here.

Chapter 5. Record structures

This chapter describes the external source format tags and attributes that define a VisualAge Generator record.

Record and Erecord definition

The RECORD tag defines attributes for individual record parts. The SQLTABLE, JOINCON, PROL, and RECDITEM tags further define aspects of a record and can have attributes.

The ERECORD tag closes the definition. If you use the RECORD tag, the ERECORD tag is required.

:RECORD tag attributes

Syntax	**Attributes**		
:record ⋮ ⋮	***:RECORD***		
:sqltable ⋮ ⋮ :joincon ⋮ ⋮	**Attributes** [ALTSPEC=	**Values** *record name*]	**Uses** Specifies an existing record whose data item structure is to be used for this record
:ejoincon ⋮ ⋮	[DATE=	'*modification date*']	Specifies the date the record was last modified
⋮ ⋮	[EDITFUNC=	*edit function*]	Specifies the function used for special data editing after all edit record items have occurred
⋮ ⋮	[EXCLUSIV=	{Y | **N**}]	Specifies that the MQSeries input queue is to be opened for exclusive use by this transaction.
:rcdhelp ⋮ ⋮ :ercdhelp	[FILENAME=	*file name*]	Associates a record specification with a physical file
:title ⋮ ⋮ :etitle	[KEY=	*item name*]	Specifies the data item that contains the record key or relative record number

Syntax	Attributes		
:	[MQGMOREC=	*record name*]	Specifies the MQSeries get options record
:	[MQMDREC=	*record name*]	Specifies the MQSeries message descriptor record
:	[MQODREC=	*record name*]	Specifies the MQSeries queue descriptor record
:	[MQOOREC=	*record name*]	Specifies the MQSeries open options record
:	[MQPMOREC=	*record name*]	Specifies the MQSeries put options record
:prol :	NAME=	*name*	Specifies the record part
:eprol :recditem : :uiprop	[NUMOCCUR=	*number of occurrences item name*]	Specifies the data item that contains the actual number of occurrences for the array that has a variable number of entries or elements
: :linkdata : :linkparm :initial :einitial :genedits :	[ORG=	{**INDEXED** | RELATIVE | SERIAL | DLISEG | SQLROW | WORKSTOR | REDEFREC | USERINTERFACE | MQMESSAGE}]	Specifies the organization of the record
:uimsgs :	[REDEFREC=	*record name*]	Specifies a record to be redefined
:	[RUNATWEB=	{Y | **N**}]	Specifies whether the record edit function is to be performed at the Web site or at the VAGen server
:numedits : :fldhelp	[SBMITVAL=	*item name*]	Specifies the item that is to contain the value of the SUBMIT button that the user pressed
: :efldhelp	[TIME=	'*modification time*']	Specifies the time the record was last modified
:	[TRANSACT=	{**Y** | N}]	Specifies that the MQSeries message is to be included in the transaction

Syntax	Attributes		
:label . . .	[USAGE=	{**SHARED** | NONSHRED}]	Specifies the default definition location for items in the record
:elabel :euiprop :vagt . . . :tracbag . . .	[VARLENTH=	*variable length item name*]	Specifies a data item in a DL/I segment that contains the length of the rest of the segment or a data item in an indexed, serial or message queue file that contains the length value for the record
:etracbag	[.]		
:erecord	***:ERECORD*** [.]		

ALTSPEC=

record name
Specifies an existing record whose data item structure is to be used for this record

ALTSPEC should not be specified for a record that is already defined as an alternate specification. In addition, an SQL row record should not be defined as an alternate specification for a record with another organization. Conversely, records not defined as SQL row records should not be specified as alternate specifications for SQL row records.

Note: If ALTSPEC is specified for any of the above records, the program will not generate.

DATE=

'modification date'
Specifies the date the record was last modified

DATE format is mm/dd/yyyy. Single quotation marks are optional.

EDITFUNC=

edit function
Specifies the name of a function used for special data editing.

EDITFUNC refers to a function that is run only in the following case:

- The value of at least one data item changed.
- All data item edits completed successfully at the location where the function is performed, whether at the WEB site or at the VAGen server.

Data item edits that run at a location other than where the edit function resides have no effect on whether the edit function is performed.

EXCLUSIV=

Y Specifies that the MQSeries input queue is to be opened for exclusive use by this transaction.

N Specifies that the MQSeries input queue is to be allowed to be shared with other transactions. (default value)

FILENAME=

file name
Associates a record specification with a physical file

FILENAME is specified for serial, relative, indexed and message queue files.

FILENAME consists of 1 to 8 characters and must meet the following conventions:

- The first character is alphabetic or national (A-Z, $, #, @).
- The remaining characters are alphanumeric or national (A-Z, 0-9, $, #, @).
- The file name cannot contain blanks or have an EZE prefix. EZEPRINT is not allowed on the FILENAME on the RECORD tag.

KEY=

item name
Specifies the data item that contains the record key for an indexed file or DL/I segment record, the relative record number for a relative file, or search field for an SQL row record

For indexed files: KEY must be defined in the data item list for the record. The key item should have the same length and record offset as the key in the records in the physical file.

For relative files: KEY does not need to be defined within the record data structure. Define the key item with a numeric (NUM), packed (PACK), or binary (BIN) data type, no decimal positions, and a maximum length of nine digits.

When a relative record file is to be accessed at execution, the key item contains a number that indicates the record position in a file relative to the beginning of the file.

For SQL: KEY specifies the name of the item in an SQL row record that is to be used as the search field in default SQL SELECT statements. The KEY specification is optional. KEY is only valid when ALTSPEC is specified.

For DL/I: KEY specifies the name of an item in a DL/I segment record that contains the segment key. The item must have the same name, length, and offset that the segment sequence field has in the DL/I database description. The key does not have a default value. Do not specify the KEY attribute if the DL/I segment has no sequence field.

The item name in a DL/I record consists of 1 to 8 characters and must meet the following conventions:

- The first character is alphabetic or national (A-Z, $, #, @)
- The remaining characters are alphanumeric or national (A-Z, 0-9, $, #, @)
- The name cannot contain blanks or have an EZE prefix.

MQGMOREC=

record name
Specifies the name of the record that contains the options to be used on a get of an MQSeries message.

MQMDREC=

record name
Specifies the name of the record that contains the MQSeries message descriptor.

MQODREC=

record name
Specifies the name of the record that contains the MQSeries object descriptor.

MQOOREC=

record name
Specifies the name of the record that contains the options to be used on an open of an MQSeries message queue.

MQPMOREC=

record name
Specifies the name of the record that contains the options to be used on a put of an MQSeries message.

NAME=

name Specifies the record part

NAME consists of 1 to 18 characters and must meet the following conventions:

- The first character is alphabetic or national (A-Z, $, #, @)
- The remaining characters are alphanumeric or national characters, hyphens, or underscores (A-Z, 0-9, $, #, @, -, _)
- The name cannot contain blanks or have an EZE prefix.

The name can be a DBCS name up to 8 DBCS characters long with no embedded blanks.

The name for a DL/I segment name can consist of 1 to 8 characters. It cannot contain underscores or hyphens, and it cannot be a DBCS name.

NUMOCCUR=

number of occurrences item name
Specifies the data item that contains the actual number of occurrences for the array that has a variable number of entries or elements

NUMOCCUR is supported only with indexed, serial and message queue records. You can specify NUMOCCUR only for variable length records. The data item NUMOCCUR has the following characteristics:

- A numeric (NUM or NUMC), binary (BIN), or packed (PACK or PACF) data type
- A length of up to 9 digits with no decimal positions
- Located in the fixed length part of the record.

The NUMOCCUR item must precede the variably occurring item in the record's item list. The variably occurring item is the last item with the highest level (lowest level number) in the record's item list.

ORG=

INDEXED
Specifies indexed organization (default value)

DLISEG
Specifies DL/I segment organization

MQMESSAGE
Specifies message queue organization

REDEFREC
Specifies redefined record organization

RELATIVE
Specifies relative organization

SERIAL
Specifies serial organization

SQLROW
Specifies SQL row organization

WORKSTOR
Specifies working storage organization

USERINTERFACE
Specifies user(U) interface(I) organization

ORG describes how the file in which the record resides is organized. The organization determines which I/O options can be used to access the record in the program.

INDEXED: Indexed organization indicates the records are in a file and are accessed by a key which is specified in the key element.

DLISEG: DL/I segment organization indicates the record is a segment in a DL/I database. The record name must be the same as the name with which the segment is defined to DL/I.

MQMESSAGE: Message queue organization indicates that the record describes an MQSeries message.

REDEFREC: A redefined record is an alternate data item structure for an existing record. The alternate data structure lets the program access the information in a record using different data item names and definitions.

Redefined records cannot be used as I/O objects, but they can be used in processing statements and as passed parameters.

RELATIVE: Relative organization indicates the file is an ordered set of fixed length records accessed by a relative record number, that is found in the key item specified for the record. For relative record access, the key item does not need to be part of the record structure. It can be in any map, record, or table used in the program.

SERIAL: Serial organization indicates the records are stored in the file in sequential order. References to the records start at the beginning and go consecutively to the end of the file.

SQLROW: SQL row organization indicates the record represents a row in a table in a relational database.

WORKSTOR: Working storage records define storage areas for temporary data items that are used in programs. The data items are not saved after execution unless they have been moved to a record and placed in a file.

Working storage can be defined as a unit of related items (data structure) in the same manner as a record. One or more single, unrelated data items can be defined for working storage instead of or in addition to the structure. Single data items are referred to as level-77 items. These items are defined with a LEVEL of 77 after all data items in the working storage structure have been defined.

USERINTERFACE: User interface records define the storage layout and the input/output properties to be used when rendering a screen definition in a JAVA client.

REDEFREC=

record name
Specifies a record to be redefined

REDEFREC identifies the name of the record that is being redefined when record organization is specified as redefined record. You can specify REDEFREC only for records with ORG=REDEFREC.

RUNATWEB=

Y Indicates that the edit function specified in the attribute EDITFUNC is to be performed at the WEB site.

N Indicates that the edit function is to be run by the VAGen server program. (default value)

SBMITVAL=

item name
Specifies the name of a data item in the UI record that will contain the value of the actual submit button on the form pressed by the user. EZEAID is the default.

SBMITVAL is supported only for UI records.

TIME=

'modification time'
Specifies the time the record was last modified

TIME format is hh:mm:ss. Single quotation marks are optional.

The following table shows the tags and attributes valid for the record organization types.

TRANSACT=

Y Specifies that the MQSeries message is to be included in the transaction. (default value)

N Specifies that the MQSeries message is not to be included in the transaction.

USAGE=

SHARED
Indicates that the definition of the item is shared with other references. The data item definition stored in the library is used. (default value)

NONSHARED
Indicates that the definition of this item is unique to this use. The data item definition stored in the library is not used.

USAGE defines the default usage for items within the record.

VARLENTH=

variable length item name
Specifies a data item in a DL/I segment that contains the length of the rest of the segment or a data item in an indexed, serial or message queue file that contains the length value for the record

For DL/I, VARLENTH is the name of a data item in a DL/I segment that contains the length value for the record. Specify the name if the segment has a variable length. You cannot specify VARLENTH if the segment has fixed length. The data item must have the same length and offset as the length field in the segment in the DL/I database description.

For DL/I, VARLENTH consists of 1 to 8 characters and must meet the following conventions:

- The first character is alphabetic or national (A-Z, $, #, @)
- The remaining characters are alphanumeric or national (A-Z, 0-9, $, #, @)
- The name cannot contain blanks or have an EZE prefix.

For indexed, serial or message queue records, VARLENTH is the name of the data item that contains the length value for the record. The VARLENTH item does not have to be within the record. If the record has fixed length, do not specify VARLENTH. VARLENTH must have the following characteristics:

- A numeric (NUM or NUMC), binary (BIN), or packed (PACK or PACF) data type
- A length of up to 9 digits with no decimal positions.

The following table shows the tags and attributes valid for the record organization types.

Tags and Attributes	INDEXED	RELATIVE	SERIAL	WORKSTOR	DLISEG	REDEFREC	SQLROW	USERINTERFACE	MQMESSAGE
:record	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**R**
ALTSPEC=	•	•	•	•	•		•	•	•
DATE=	•	•	•	•	•	•	•	•	•
EDITFUNC=								•	
EXCLUSIV=									•
FILENAME=	**R**	**R**	**R**						**R**
KEY=	**R**	**R**			•		•		
MQGMOREC=									•
MQMDREC=									•
MQODREC=									•
MQGOOREC=									•
MQPMOREC=									•
NAME=	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**R**
NUMOCCUR=	•		•						•
ORG=	•	•	•	•	•	•	•	•	•
REDEFREC=						**R**			
RUNATWEB=								•	
SBMITVAL=								•	
TIME=	•	•	•	•	•	•	•	•	•
TRANSACT=									•
USAGE=	•	•	•	•	•	•	•	•	•
VARLENTH=	•		•		•				•
:sqltable							•		
LABEL=							•		
TABLEID=							**R**		
TBLNHVAR=							•		
:joincon							•		

Tags and Attributes	INDEXED	RELATIVE	SERIAL	WORKSTOR	DLISEG	REDEFREC	SQLROW	USERINTERFACE	MQMESSAGE
HOSTVAR=							•		
:ejoincon							•		
:rcdhelp								•	
:ercdhelp								•	
:title								•	
:etitle								•	
:prol	•	•	•	•	•	•	•	•	•
:eprol	•	•	•	•	•	•	•	•	•
:recditem	•	•	•	•	•	•	•	•	•
BYTES=	•	•	•	•	•	•	•	•	•
COLNAME=							•		
DATACODE=							•		
DECIMALS=	•	•	•	•	•	•	•	•	•
DESC=	•	•	•	•	•	•	•	•	•
EVENSQL=	•	•	•	•	•	•	•	•	•
KEY=							•		
LEVEL=	•	•	•	•	•	•		•	•
NAME=	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**R**
OCCURS=	•	•	•	•	•	•		•	•
READONLY=							•		
TYPE=	•	•	•	•	•	•	•	•	•
USAGE=	•	•	•	•	•	•	•	•	•
:uiprop								•	
EDITORDR=								•	
UITYPE=								•	
OCCURSFR=								•	
SELINDEX=								•	
:linkdata								•	
FIRSTUI=								•	

Tags and Attributes	INDEXED	RELATIVE	SERIAL	WORKSTOR	DLISEG	REDEFREC	SQLROW	USERINTERFACE	MQMESSAGE
NEWWIN=								•	
PROGRAM=								•	
:linkparm								•	
NAME=								•	
VALUEITM=								•	
:initial								•	
:einitial								•	
:genedits								•	
EDITFUNC=								•	
EDITTBLE=								•	
EDITTYPE=								•	
FILLCHAR=								•	
FLDFOLD=								•	
INPUTREQ=								•	
MININPUT=								•	
RUNATWEB=								•	
SOSI=								•	
:uimsgs								•	
FUNCKEY=								•	
MININKEY=								•	
RANGEKEY=								•	
REQKEY=								•	
TBLEKEY=								•	
TYPEKEY=								•	
:numedits								•	
CURRENCY=								•	
CURRSYMB=								•	
NUMSEP=								•	
RANGE=								•	

Tags and Attributes	INDEXED	RELATIVE	SERIAL	WORKSTOR	DLISEG	REDEFREC	SQLROW	USERINTERFACE	MQMESSAGE
SIGN=								•	
ZEROEDIT=								•	
:fldhelp								•	
:efldhelp								•	
:label								•	
:elabel								•	
:euiprop								•	
:erecord	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**R**

Note:
• indicates the tags and attributes that are valid.
R indicates the tag or attribute is required.

SQL table name definition

The SQLTABLE tag specifies the name of a relational table that an SQL row record represents. The SQLTABLE tag is required for an SQL row record unless there is an alternate specification record specified (altspec). This optional tag further describes the RECORD and ERECORD tag set.

Repeat the SQLTABLE tag for each table represented by the record.

:SQLTABLE tag attributes

Syntax	**Attributes**		
:record ⋮	***:SQLTABLE***		
:sqltable ⋮	**Attributes** [LABEL=	**Values** *'table label'*]	**Uses** Specifies a shortened version of the table name
:joincon	TABLEID=	*'table ID'*	Identifies the table represented by the SQL row record

Syntax	Attributes		
: :	[TBLNHVAR=	{Y | **N**}]	Identifies the TABLEID value is a table name host variable
:ejoincon : :	[.]		
:erecord			

LABEL=

'table label'
Specifies a shortened version of the table name

Use LABEL as a qualifier to uniquely identify column names in SQL row definitions and SQL statements when the SQL row record represents two or more tables joined together. The maximum length of a table label is 4 bytes.

Import automatically generates a table label if the label is not specified.

TABLEID=

'table ID'
Identifies the table represented by the SQL row record

The format of the table ID is not checked except under the conditions specified in the note below. The table ID can be any name that is coded in a FROM clause in an SQL SELECT statement. The maximum length of the table ID is 60 bytes. The table ID name is inserted in SQL statements and passed to the database manager exactly as specified.

TBLNHVAR=

Y Specifies that the TABLEID value is a table name in host variable format

N Specifies that the TABLEID value is not a table name in host variable format (default value)

Note: If TBLNHVAR is set to Y, and the TABLEID value begins with a ?, the TABLEID value is treated as a table name host variable, and the format

is checked. If TBLNHVAR is set to Y, but the TABLEID value does not begin with a ?, the table identified by TABLEID will not be treated as a table name host variable.

Default selection criteria definition

The JOINCON tag defines the default selection criteria used when SELECT statements are built for functions that have an SQL row record as the I/O object. An SQL join permits data retrieval from two or more tables based on matching columns, and a join condition specifies a relationship between the tables to be joined.

You can specify JOINCON only if you have specified values for TABLEID on the SQLTABLE tag or ALTSPEC on the RECORD tag. The JOINCON tag is optional and further describes the RECORD and ERECORD tag set. The EJOINCON tag closes the default selection criteria, and if you use the JOINCON tag, the EJOINCON tag is required.

:JOINCON tag attributes

Syntax	Attributes		
:record ⋮	***:JOINCON***		
:sqltable ⋮	**Attributes** [HOSTVAR=	**Values** {'?' | '@'}]	**Uses** Specifies the character used within the clause text to indicate the beginning of a host variable name
:joincon ⋮	[.[*clause text*]] ***:EJOINCON*** [.]		Contains the WHERE clause of the join condition
:ejoincon ⋮			
:erecord			

HOSTVAR=

'?'	Specifies a ? is used to begin host variable names
'@'	Specifies an @ is used to begin host variable names (default value)

Export always uses a ? as the host variable name indicator so that an @ can be properly interpreted as a valid character in a name. Using the @ as the host variable name indicator, when variable names also contain the @, is ambiguous and can cause unpredictable results on import.

clause text

clause text
Specifies the clause for the SQL statement

CLAUSE TEXT is the WHERE clause of the join condition.

Note: 'WHERE' is not included in the WHERE clause.

Within the clause text, SQL column names always begin with the exclamation point (!). When produced from VisualAge Generator export, host variable names always begin with the question mark (?).

Statements can consist of multiple lines of text. A statement line cannot exceed 2000 bytes.

Record help definition

The RCDHELP tag begins the set of text lines that are to be displayed as help for this record. This optional tag further defines the RECORD and ERECORD tag set. The ERCDHELP tag closes the definition, and if you use the RCDHELP tag, the ERCDHELP tag is required. The RCDHELP and ERCDHELP tags are only allowed on user interface records.

The RCDHELP tag can appear only once within a record definition.

:RCDHELP tag values

Syntax	**Usage**	
:record . .	***:RCDHELP***	
:rcdhelp . .	**Values** [.[*help text*]]	**Uses** Specifies the help text for a UI record
:ercdhelp . .	***:ERCDHELP*** [.]	
:erecord		

Only the library size and machine size limit the number of lines that you can specify. Each line can contain up to 2000 bytes. You can use both uppercase and lowercase characters. The help text must be preceded by a period (.). Text is saved as entered.

Record title definition

The TITLE tag begins the default text line that is to be displayed as the title for this record. This optional tag further defines the RECORD and ERECORD tag set. The ETITLE tag closes the definition, and if you use the TITLE tag, the ETITLE tag is required. The TITLE and ETITLE tags are only allowed on user interface records.

The TITLE tag can appear only once within a record definition.

:TITLE tag values

Syntax	**Usage**	
:record ⋮ ⋮	***:TITLE***	
:title ⋮ ⋮	**Values** [.[*title text*]]	**Uses** Specifies the title for a UI record
:etitle ⋮ ⋮	***:ETITLE*** [.]	
:erecord		

Only one line of title text is allowed. This line can contain up to 2000 bytes. You can use both uppercase and lowercase characters. The title text must be preceded by a period (.). Text is saved as entered.

Record prologue definition

The PROL tag is a text description of a defined record. This optional tag further defines the RECORD and ERECORD tag set. The EPROL tag closes the description, and if you use the PROL tag, the EPROL tag is required.

:PROL tag values

Syntax	**Usage**	
:record		

Syntax	Usage	
. .	*:PROL*	
:prol . .	**Values** [.[*prologue lines*]]	**Uses** Describes the record
:eprol . .	*:EPROL* [.]	
:erecord		

Only the library size and machine size limit the number of lines that you can specify. Each line can contain up to 60 bytes. You can use both uppercase and lowercase in the prologue. The prologue must be preceded by a period (.). Text is saved as entered. When text is longer than 60 characters, it is split into multiple lines during import.

Record item definition

The RECDITEM tag is a list of data items to be defined for this record. This optional tag with its attributes further defines the RECORD and ERECORD tag set.

Repeat the RECDITEM tag for each data item in the record except when it is an alternate specification record. Specify the TYPE, BYTES, DECIMALS, EVENSQL, and DESC attributes for nonshared data items only.

Define the shared data items in the list as separate parts using the ITEM tag. The order of the RECDITEM tags determines the order of the data items within the record.

:RECDITEM tag attributes

Syntax	Attributes		
:record . .	*:RECDITEM*		
:recditem . .	**Attributes** [BYTES=	**Values** *field length in bytes*]	**Uses** Specifies the field length in bytes for the data item based on the data item type

Syntax	Attributes		
:erecord	[COLNAME=	*'column name'*]	Specifies the column name that identifies an item in an SQL row record to the relational database manager
	[DATACODE=	*SQL data code*]	Specifies the SQL data type of the data item in an SQL row record to the relational database manager
	[DECIMALS=	{**0** | *decimal places*}]	Specifies the number of decimal places
	[DESC=	*'field description'*]	Describes a data item
	[EVENSQL=	{*Y* | **N**}]	Specifies whether packed fields of even length or odd length are passed to a relational database
	[KEY=	{*Y* | **N**}]	Specifies whether the data item is a key column for the SQL table
	[LEVEL=	{**10** | *number*}]	Indicates relative placement of a data item to adjacent data items
	NAME=	{* | *item name*}	Specifies a data item name
	[OCCURS=	{**1** | *number of occurrences*}]	Specifies the number of repetitions of a data item in a data structure
	[READONLY=	{Y | N}]	Indicates whether an item in an SQL row record can be written to the relational database
	[TYPE=	{BIN | **CHA** | DBCS | HEX | MIX | NUM | NUMC | PACK | PACF | UNICODE}]	Specifies the data item type for nonshared data items
	[USAGE=	SHARED | NONSHARED]	Specifies the location of the item definition
	[.]		

BYTES=

field length in bytes
Specifies the field length in bytes for the data item based on the data item type.

3 Default for CHA, MIX, NUM and NUMC

2 Default for BIN, PACK, PACF, HEX

6 Default for DBCS and UNICODE

BYTES is the number of bytes required to store a data item value internally. The following table lists the valid BYTES values by data type:

Item Type	Byte Values
CHA, HEX, MIX	1-32,767
DBCS, UNICODE	2-32,766 (even only)
NUM, NUMC	1-18
PACK, PACF	1-10
BIN	2, 4, 8 only
BIN in SQL row record	2, 4 only

Note: The BYTES attribute can be specified only for nonshared data items on the RECDITEM tag.

COLNAME=

'column name'
Specifies the column name that identifies an item in an SQL row record to the relational database manager

COLNAME can be up to 36 characters long and can be either of the following:

- The actual name of a column in the relational table or view definition.

  If the SQL row record is defined as a join of multiple tables or views, the column name should be qualified by the table label defined for the table or view to which it belongs.
- An SQL expression made up of column names and SQL operators, constants, and built-in functions.

As an expression is entered into the SQL column name field, a virtual column is defined which can be used as a read-only data item in the SQL row record definition. The expression is calculated at the time the SQL row is read from the database.

The specified name is not checked for validity. All single-byte characters not within double quotation marks are folded to uppercase. The name is inserted into generated SQL statements as entered. The name is validated by the relational database manager during SQL statement preparation for a program.

DATACODE=

SQL data code
Identifies the SQL data type of the data item to the relational database manager

VisualAge Generator determines the SQL data codes for all data items except HEX data items. You must enter the data code for HEX items. The DATACODE attribute is produced on export for HEX, CHA, DBCS, and UNICODE data items only.

DECIMALS=

0 Specifies no decimal places (default value)

decimal places
Specifies the number of decimal places

DECIMALS specifies the number of decimal places in numeric data items. This number is included in a data item's length; however, the decimal point is not stored with the data. The maximum number of decimal places is 18 or the number of digits defined for the data item, whichever is smaller. The default is 0 for no decimal places.

Note: The DECIMALS attribute can be specified only for nonshared data items on the RECDITEM tag.

DESC=

'field description'
Describes a data item

DESC consists of 1 to 30 characters that can be entered in uppercase and lowercase. Mixed data is also permitted.

Note: The DESC attribute can be specified only for nonshared data items on the RECDITEM tag.

EVENSQL=

Y Specifies whether packed fields of even length are passed to a relational database

N Specifies whether packed fields of odd length are passed to a relational database (default value)

EVENSQL indicates whether VisualAge Generator passes even- or odd-length packed fields to relational databases. Specify EVENSQL as Y only if you are using an SQL table that has even-length columns defined.

Note: The EVENSQL attribute can be specified only for nonshared data items on the RECDITEM tag.

KEY=

Y Specifies that a data item is a key column for the SQL table

N Specifies that a data item is not a key column for the SQL table (default value)

KEY specifies that the data item is a key column for the SQL table. One use of KEY is when the SQL table is defined with a multiple-column key. This enables VisualAge Generator to provide default SQL statements that use the indexes in the SQL table correctly.

Note: KEY is used only with SQL row records.

LEVEL=

10 Default value

number
Specifies the relative placement of a data item to adjacent data items in a data structure

LEVEL is a number (3-49, or 77 for single data items in working storage) that is unique to a record, table, or working storage definition. Level numbers can differ for the same data item that is used in several record, table, or working storage definitions.

Data items with the lowest level number in a structure occupy the highest position in the structure. Data items with higher level numbers represent substructures of the previous item in the structure list with a lower level number.

The total length of the data items in a substructure must equal the length of the owning data item.

Working storage can contain single data items in addition to or in place of a data structure. Single data items have a level specified as 77 and must follow the structure if one exists.

NAME=

***** Specifies a filler data item

item name
Specifies the data item part

For a DL/I item, NAME consists of 1 to 8 characters and meets the following conventions:

- The first character is alphabetic or national (A-Z, $, #, @)
- The remaining characters are alphanumeric or national (A-Z, 0-9, $, #, @)
- The name cannot contain blanks or have an EZE prefix.

For a non-DL/I item, NAME consists of 1 to 32 characters and must meet the following conventions:

- The first character is alphabetic or national (A-Z, $, #, @)
- The remaining characters are alphanumeric or national characters, hyphens, or underscores (A-Z, 0-9, $, #, @, -, _)
- The name cannot contain blanks or have an EZE prefix.

For a non-DL/I item, the name can be a DBCS name up to 15 DBCS characters long with no embedded blanks.

A data item name in a record structure can be specified with an asterisk. This type of data item, called a filler, cannot be referred to by the program; it acts as a space holder.

Note: SQL row records cannot contain a filler data item. A filler is always a nonshared data item.

OCCURS=

1 Default data item OCCURS number

number of occurrences
Specifies the number of repetitions of a data item in a data structure

You can specify OCCURS as a number from 1 to 32,767. Data items cannot have nested occurrences. After you define a data item with OCCURS greater than 1, no data item within its substructure can have an OCCURS value greater than 1.

The OCCURS characteristic for a data item applies only for the record in which the item is defined. If you use a data item in more than one record, the OCCURS can be different in each.

READONLY=

Y Indicates an item in an SQL row record cannot be written to the relational database

N Indicates an item in an SQL row record can be written to the relational database

READONLY determines what columns are included in generated SQL statements that write to the database. Specify READONLY=Y for columns that cannot be updated and for columns that the program never needs to change. In addition, the following items must be read only:

- Items with SQL column names that are expressions (not valid SQL column names)
- Items that come from an SQL row defined as a join.

TYPE=

BIN Binary number

CHA Character data (default value)

DBCS Double-byte character data

HEX Hexadecimal data

MIX DBCS data intermingled with single-byte character data

NUM Numeric characters with positive sign in F format

NUMC
Numeric characters with positive sign in C format

PACF Packed decimal characters with positive sign in F format

PACK Packed decimal characters with positive sign in C format

UNICODE
Unicode character data (only valid for Developer on JAVA)

TYPE specifies the internal format or type of data. The data type determines how the item is processed when referred to in processing statements.

Note: The TYPE attribute can be specified for nonshared data items only on the RECDITEM item tag. MIX, NUM, NUMC, and PACF values are not allowed in SQL row records.

USAGE=

SHARED
Indicates that the definition of the item is shared with other references. The definition stored in the library is used.

NONSHARED
Indicates that the definition of the item is unique to this use. The definition stored in the library is not used.

This USAGE value overrides the value specified on the USAGE attribute of the :RECORD tag. If a USAGE value is not specified, the value specified on the USAGE attribute of the :RECORD tag is the default.

User interface properties definition

The UIPROP tag specifies property information about a field in a user interface record. This optional tag further defines the RECDITEM tag. The optional tags LINKDATA, LINKPARM, GENEDITS, UIMSGS, NUMEDITS, FLDHELP, and LABEL provide additional user interface information about the field and are only allowed for nonshared data items within UI records. The EUIPROP tag closes the definition and is required for each UIPROP tag.

Syntax	**Attributes**		
:record : :	***:UIPROP***		
	Attributes	**Values**	**Uses**
:uiprop	[EDITORDR=	*number*]	Specifies the position of the field in the edit sequence
: :	[OCCURSFR=	*item name*]	Specifies the name of the item that contains the number of occurrences for an array item.
:euiprop	[SELINDEX=	*item name*]	Specifies the name of the item that contains the selection index for an array item
: :	[UITYPE=	FORM | HIDDEN | INPUT | INPUTOUTPUT | NONE | **OUTPUT** | PROGRAMLINK | SUBMIT | SUBMITBYPASS]	Specifies how the item is used in the user interface and together with the other attributes helps determine the default UI elements used to implement the HTML form
:erecord	[.] ***:EUIPROP*** [.]		

EDITORDR=

number
Specifies the position of the field in the edit sequence

EDITORDR allows you to modify the sequence in which user interface fields are checked on input to a program. It is allowed only for items with a UITYPE of INPUT or INPUTOUTPUT. If you specify EDITORDR for any item in the record, you must specify it for all INPUT or INPUTOUTPUT fields. If not specified for any item in the record, the default edit order is the order in which the items appear in the record.

OCCURSFR=

item name
Specifies the name of the item that contains the number of occurrences for an array item

The values set into this item determine the number of occurs in the array item that actually get displayed in a list. The item named must be a numeric data item with no decimal places.

SELINDEX=

item name
Specifies the name of the item that contains the selection index for an array item

The values set into this item are the indices of the elements that were selected by the user. If the item named is a single element, the list will be a single select list. If the item is an array, the list will be a multiple select list.

UITYPE=

FORM
Indicates that the item will contain a value or set of values that can be received into the Submit Value Item when the user submits a form back to the server. No other data is sent back to the server. This is identical to PROGRAMLINK except that items that are substructured under an item with this UITYPE are considered to be part of the Form.

HIDDEN
Indicates that the field is not intended to show on the end user page. However, the data of these fields is passed when a form containing a hidden field is submitted.

INPUT
Indicates that the field is intended to be an input field. INPUT is not allowed for superstructure items.

INPUTOUTPUT
Indicates that the field is intended to be used for both input and output. INPUTOUTPUT is not allowed for superstructure items.

NONE
Indicates that the field is not intended to be displayed.

OUTPUT
Indicates that the field is intended to be an output field. (default value)

PROGRAMLINK
Indicates that the item will contain a value or set of values that can be received into the Submit Value Item when the user submits a form back to the server. No other data is sent back to the server.

SUBMIT
Indicates that the item will contain a value or set of values that can be received into the Submit Value Item when the user submits a form back to the server. Other data from the form is sent back to the server.

SUBMITBYPASS
Same as SUBMIT except that input edits are bypassed at runtime.

Link properties definition

The LINKDATA tag defines user interface properties that can be specified for an item in a record, named on a RECDITEM tag, that has a UITYPE of FORM or PROGRAMLINK. This optional tag further defines the UIPROP and EUIPROP tag set.

:LINKDATA tag attributes

Syntax	**Attributes**		
:record ⋮	***:LINKDATA***		
:uiprop ⋮	**Attributes** [FIRSTUI=	**Values** *record name*]	**Uses** Specifies the name of a user interface record
:linkdata	[NEWWIN=	{Y | **N**}]	Specifies whether or not a new window is to be created to present this information
:euiprop ⋮	PROGRAM =	*program name*	Specifies the name of a program that is expected to be invoked when this UI record is used on a transfer.
:erecord	[.]		

FIRSTUI=

record name
Specifies the name of a user interface record

NEWWIN=

Y Specifies that a new window is to be created to present this information

N Specifies that a new window will not be created to present this information (default value)

PROGRAM=

program name
Specifies the name of a program that is expected to be invoked when this UI record is used on a transfer

Link parameter definition

The LINKPARM tags define user interface properties that can be specified for an item in a record, named on a RECDITEM tag, that has a UITYPE of FORM or PROGRAMLINK. This optional tag further defines the UIPROP and EUIPROP tag set.

The LINKPARM tag cannot be specified without the LINKDATA tag. Repeat the LINKPARM tag for each data item that is to be a parameter when the UI record is passed on an XFER statement. The order of the LINKPARM tags determines the order of the parameters.

:LINKPARM tag attributes

Syntax	**Attributes**		
:record . .	***:LINKPARM***		
:uiprop . .	**Attributes**	**Values**	**Uses**
:linkparm	NAME=	*item name*	Specifies the name of the item that is a parameter to the record bean
. .	[VALUEITM=	{*item name* | *'literal'*}]	Specifies the value that is to be used for the named parameter
:euiprop . .	[.]		
:erecord			

NAME=

item name
Specifies the name of the item that is a parameter to the record bean

VALUEITM=

item name
Specifies the name of the item containing the value that is to be used for the named parameter

'literal'
Specifies a literal value that is to be used for the named parameter. Character, Hex, DBCS, or Mixed strings must be enclosed in single quotes.

Initial definition

The INITIAL tag begins the text lines that contain the initial value for a field. This optional tag further defines the UIPROP and EUIPROP tag set. The EINITIAL tag closes the definition, and if you use the INITIAL tag, the EINITIAL tag is required. The INITIAL and EINITIAL tags are allowed only on user interface records.

The INITIAL tag can appear only once for each item within a record definition.

:INITIAL tag values

Syntax	Usage	
:record . .	*:INITIAL*	
:uiprop		
:initial . .	**Values** [.[*initial value*]]	**Uses** Specifies the initial value for a field in a UI record
:einitial . .	*:EINITIAL* [.]	
:euiprop		
:erecord	[.]	

The initial value is a single line that can contain up to 2000 bytes. You can use both uppercase and lowercase characters. The initial value must be preceded by a period (.). Text is saved as entered.

Initial value definition

The INITIAL tag begins the set of text lines that define the initial value for a field or button in a user interface record. This optional tag further defines the RECORD and ERECORD tag set. The EINITIAL tag closes the definition, and if you use the INITIAL tag, the EINITIAL tag is required. The INITIAL and EINITIAL tags are only allowed on user interface records.

The INITIAL tag can appear only once within a record definition.

:INITIAL tag values

Syntax	**Usage**	
:record . . .	***:INITIAL***	
:rcdhelp . . .	**Values** [.[*help text*]]	**Uses** Specifies the help text for a UI record
:ercdhelp . . .	***:EINITIAL*** [.]	
:erecord		

Only the library size and machine size limit the number of lines that you can specify. Each line can contain up to 2000 bytes. You can use both uppercase and lowercase characters. The initial value must be preceded by a period (.). Text is saved as entered.

General edit characteristic definition

The GENEDITS tag defines a list of general editing characteristics that can be specified for items in a UI record. This optional tag further defines the UIPROP and EUIPROP tag set.

:GENEDITS tag attributes

Syntax	**Attributes**		
:record . . .	***:GENEDITS***		
:uiprop . . .	**Attributes** [EDITFUNC=	**Values** *function name*]	**Uses** Specifies the function used for special data editing

Syntax	Attributes		
:genedits	[EDITTBLE=	*table name*]	Specifies the table used for special data editing
⋮	[EDITTYPE=	{BOOLEAN | DATE | TIME | ***NONE***}]	Specifies the type of editing to be done on the data item
:euiprop	[FILLCHAR=	*'fill character'*]	Specifies the character used to fill unused field positions
⋮	[FLDFOLD=	{Y | **N**}]	Specifies whether data entered into the field is folded
:erecord	[INPUTREQ=	{Y | **N**}]	Specifies whether valid data must be entered
	[MININPUT=	{**N** | *positions*}]	Specifies the minimum number of required characters
	[RUNATWEB=	{Y | **N**}]	Specifies whether edits are to be performed at the WEB site or at the VG Server
	[SOSI=	{Y | N}]	Specifies whether to check mixed data to see if it will fit into the field after conversion.
	[.]		

EDITFUNC=

function name
Specifies the name of a function used for special editing of data in a variable field

EDITFUNC indicates the name of a routine to be used for special editing of data that the user enters in a variable field. You can specify one of the following:

- The name of one of the following special funtion word subroutines:
  - Modulus 10 check digit routine (EZEC10)
  - Modulus 11 check digit routine (EZEC11)
- The name of an edit routine.

EDITTBLE=

table name
Specifies the name of a table used for special editing of data in a variable field

EDITTYPE=

BOOLEAN
Specifies to the UI record that the data in the item is in the form of a Boolean value.

DATE Specifies to the UI record that the data in the item is in the form of a Date value.

TIME Specifies to the UI record that the data in the item is in the form of a Time value.

NONE
Specifies to the UI record that the data in the item is in no particular form. (default value)

FILLCHAR=

'fill character'
Specifies the character used to fill unused field positions

FILLCHAR indicates the character used to fill unused field positions on output.

FLDFOLD

Y Specifies that data is folded

N Specifies that data is not folded (default value)

FLDFOLD specifies whether lowercase alphabetic characters entered by the user are to be folded (converted) to uppercase.

INPUTREQ=

Y Specifies that data must be entered in the field

N Specifies that input is not required in the field (default value)

INPUTREQ indicates whether data must be entered in a field. When a field contains data other than blanks for character type or zeros for numeric types, the field is considered to have input. Blanks in a character field and zero in a numeric field do not satisfy the input required edit check. Even if blanks and zeros are valid values, INPUTREQ returns an error message unless the user types data into the field.

MININPUT=

N Specifies no minimum number of characters required (default value)

positions
Specifies the minimum number of characters that must be entered in a valid variable field

MININPUT specifies the minimum number of characters that must be entered in a variable field if any data is entered.

RUNATWEB=

Y Specifies that the edits are to be performed at the WEB Server site

N Specifies that the edits are to be performed by the VG Server program

SOSI=

Y Specifies that mixed data entered in the field on an ASCII device is to be checked to see if it will fit into the field after conversion into the EBCDIC SO/SI format for mixed data (default for mixed type fields)

N Specifies that mixed data entered in the field on an ASCII device is not to be checked to see if it will fit into the field after conversion into the EBCDIC SO/SI format for mixed data (default for all fields except mixed type fields)

Message key definition

The UIMSGS tag specifies property information about message keys for a field in a user interface record. This optional tag further defines the UIPROP and EUIPROP tag set.

:UIMSGS tag attributes

Syntax	**Attributes**		
:record : :	*:UIMSGS*		
:uiprop : :	**Attributes**	**Values**	**Uses**
	[FUNCKEY=	*'edit function message key'*]	Specifies the message key to be used when an edit function is used to determine an error condition exists
:uimsgs	[MININKEY=	*'minimum input message key'*]	Specifies the message key to be used when fewer than the minimum characters are input
: : :	[RANGEKEY=	*'numeric range message key'*]	Specifies the message key to be used when the user enters data that is out of range for this field
:euiprop	[REQKEY=	*'input required message key'*]	Specifies the message key to be used when the user does not enter data into a field that must contain data

Syntax	Attributes		
:			
:	[TBLEKEY=	*'edit table message key'*]	Specifies the message key to be used when an edit table is used to determine an error condition exists
	[TYPEKEY=	*'data type message key'*]	Specifies the message key to be used when the type of data entered by the user is not valid for the field
:erecord	[.]		

FUNCKEY=

'edit function message key'
Specifies the message key to be used when an edit function is used to determine an error condition exists

MININKEY=

'minimum input message key'
Specifies the message key to be used when fewer than the minimum characters are input

RANGEKEY=

'numeric range message key'
Specifies the message key to be used when the user enters data that is out of range for this field

REQKEY=

'input required message key'
Specifies the message key to be used when the user does not enter data into a field that must contain data

TBLEKEY =

'edit table message key'
Specifies the message key to be used when an edit table is used to determine an error condition exists

TYPEKEY=

'data type message key'
Specifies the message key to be used when the type of data entered by the user is not valid for the field

Numeric edit characteristic definition

The NUMEDITS tag defines a list of numeric editing characteristics that can be specified for items in a UI record. This optional tag further defines the UIPROP and EUIPROP tag set. It can only be specified for numeric data items.

:NUMEDITS tag attributes

Syntax	**Attributes**		
:record . .	***:NUMEDITS***		
:uiprop	**Attributes**	**Values**	**Uses**
.	CURRENCY=	{Y | **N**}]	Specifies whether a default currency symbol is to be used
	[CURRSYMB=	'*symbol*']	Specifies a 1- to 3-character currency symbol
:numedits			
.	[NUMSEP=	{Y | **N**}]	Specifies whether data can contain numeric separators
	[RANGE=	lowvalue highvalue]	Specifies the range of valid numeric values
:uiprop			
.	[SIGN=	{LEA | TRA | N}]	Specifies a sign in the field as leading, trailing, or none
	[ZEROEDIT=	{Y | **N**}]	Specifies the format for fields that have zero values
:erecord	[.]		

CURRENCY=

Y Specifies that the currency symbol is to be used

N Specifies that no currency symbol is to be used (default value)

CURRENCY indicates whether the currency symbol is supported in the field. Y is valid only for numeric fields. When defining field length, there must be enough room for each character specified on CURRSYMB.

CURRSYMB=

'symbol'
Specifies a 1- to 3-character currency symbol to be used if CURRENCY is Y

NUMSEP=

Y Specifies that a numeric separator is to be used with the field

N Specifies that a numeric separator is not to be used with the field (default value)

NUMSEP specifies whether data in a field can contain the previously defined numeric separator. The default numeric separator is a comma. Your system administrator can change the default character using the customization procedures for language-dependent options.

If YES is specified, numeric separators are allowed in the field on input. After input editing, the numeric separators are removed before the field is placed in internal storage. On output, numeric separators are inserted between every three significant digits. Every fourth position to the left of the decimal point will be a separator. You can only specify NUMSEP=Y for numeric fields. When defining field length, remember that each numeric separator takes up one position. If NUMSEP=N is specified, validation is done to ensure that the user did not enter a numeric separator in the field.

RANGE=

lowvalue
Specifies the smallest numeric value for a field

highvalue
Specifies the largest numeric value for a field

RANGE specifies the range of valid numeric values for a field. Lowvalue is the smallest numeric value that can be entered in a specified field. Highvalue is the largest numeric value that can be entered in a specified field. The high and low values must have the same length, number of decimal positions, and sign as defined for the field. Both low and high values must be specified if this attribute is used. The values are separated by a space and cannot be longer than the length of the field being defined. RANGE can be specified only for numeric fields.

SIGN=

LEA	Specifies a leading sign
TRA	Specifies a trailing sign
N	Specifies no sign (default value)

SIGN specifies whether a sign is displayed in a field and whether it is a leading or a trailing sign. You can specify signs for numeric fields only. When defining field length, remember the sign takes up one position.

ZEROEDIT=

Y	Specifies the display format for numeric fields containing 0 values
N	Specifies that no editing is done on 0 numeric fields (default value)

ZEROEDIT specifies the display format for numeric fields that have 0 values.

Field help definition

The FLDHELP tag begins the set of text lines that are to be displayed as help for this field. This optional tag further defines the UIPROP and EUIPROP tag set. The EFLDHELP tag closes the definition, and if you use the FLDHELP tag, the EFLDHELP tag is required. The FLDHELP and EFLDHELP tags are only allowed on user interface records.

The FLDHELP tag can appear only once for each item within a record definition.

:FLDHELP tag values

Syntax	Usage	
:record . . .	***:FLDHELP***	
:uiprop . . .	**Values** [.[*field help text*]]	**Uses** Specifies the help text for a field in a UI record
:fldhelp . . .	***:EFLDHELP*** [.]	
:efldhelp :erecord	[.]	

Only the library size and machine size limit the number of lines that you can specify. Each line can contain up to 2000 bytes. You can use both uppercase and lowercase characters. The help text must be preceded by a period (.). Text is saved as entered.

Label definition

The LABEL tag begins the text lines that are to be displayed as the label text for this field. This optional tag further defines the UIPROP and EUIPROP tag set. The ELABEL tag closes the definition, and if you use the LABEL tag, the ELABEL tag is required. The LABEL and ELABEL tags are only allowed on user interface records.

The LABEL tag can appear only once for each item within a record definition.

:LABEL tag values

Syntax	Usage	
:record		

Syntax	Usage	
. . .	*:LABEL*	
:label . . .	**Values** [.[*label text*]]	**Uses** Specifies the label for an item in a UI record
:elabel . . .	*:ELABEL* [.]	
:erecord	[.]	

If the item is an array, one label is allowed for each element in the array. Otherwise, only one label is allowed. Each label must be contained on a single line. A single line cannot exceed 2000 bytes. You can use both uppercase and lowercase characters. The label text must be preceded by a period (.). Text is saved as entered.

Templates tracability information

Templates tracability information is defined for a record part as described in "Chapter 16. Templates traceability information structures" on page 213.

Record definition syntax example

```
:record    name     = xxxxxxxxxxxxxxxxx  date = 'mm/dd/yyyy'
           time     = 'hh:mm:ss'    org  = xxxxxxxx
           altspec  = xxxxxxxxxxxxxxxxx  filename = xxxxxxxx
           key      = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
           redefrec = xxxxxxxxxxxxxxxxx
           varlenth = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
           numoccur = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
           usage    = xxxxxxxxx
           sbmitval = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.
:sqltable  tableid  = 'Enter up to 60 characters'
           tblnhvar = x
           label    = 'xxxx'.
:joincon   hostvar  = 'x'.
           EMPNO IN (SELECT EMPNO FROM DSN8130.TEMPL WHERE
           WORKDEPT = 'E11');
:ejoincon.
:rcdhelp.
This text is the help text for the record.
:ercdhelp.
:title.
This text is the default title for the UI record.
:etitle.
:prol.
Each prologue line can be up to 60 characters long.  IMPORT
```

```
will split lines longer than 60 characters.
:eprol.
:recditem name     = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
          level    = xx                  occurs = xxxxx
          usage    = xxxxxxxxx
          type     = xxxx                bytes  = xxxxx
          colname  = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
          readonly = x          datacode = xxx
          decimals = xx         evensql  = x
          desc     = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx'
          key      = x.
:uiprop    uitype    = xxxxxxxxxxx
           editordr  = xxxxx
           occursfr  = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
           selindex  = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.
:linkdata  program   = xxxxxxx
           firstui   = xxxxxxxxxxxxxxxxx
           newwin    = x.
:linkparm  name      = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
           valueitm  = 'xxx'.
:genedits  edittype  = xxxx
           editfunc  = xxxxxxxxxxxxxxxxx
           edittble  = xxxxxxx
           fillchar  = 'x'
           fldfold   = x
           inputreq  = x
           mininput  = xxxxx
           runatweb  = x
           sosi      = x.
:uimsgs    tblekey   = 'xxxxxxxxxx'
           funckey   = 'xxxxxxxxxx'
           mininkey  = 'xxxxxxxxxx'
           rangekey  = 'xxxxxxxxxx'
           reqkey    = 'xxxxxxxxxx'
           typekey   = 'xxxxxxxxxx'.
:numedits  currency  = x    sign = xxx     numsep = x
           currsymb  = 'xxx'
           range     = xxxxxxxxxxxxxxxxxx    xxxxxxxxxxxxxxxxxx
           zeroedit  = x.
:fldhelp.
Field help text goes here
:efldhelp.
:label.
Field label text goes here
:elabel.
:euiprop.
:erecord.
```

Note: This example illustrates the syntax of all record definition tags and attributes. Actual definitions do not require or use all tags and attributes shown here.

Chapter 6. Table structures

This chapter describes the external source format tags and attributes that define a VisualAge Generator table.

Table definition

The TBLE tag and its attributes define table parts. The TBLE tag specifies data that can be used for the following:

- Editing data that a user enters on a map
- Storing information that a program refers to during execution.

The GENOPTS, PROL, DEFITEM, CONTITEM, and ROW tags further define aspects of a table part and can have attributes. The ETBLE tag closes the definition, and if you use the TBLE tag, the ETBLE tag is required.

:TBLE tag attributes

Syntax	**Attributes**		
:tble ⋮ ⋮	***:TBLE***		
:genopts ⋮ ⋮	**Attributes** [DATE=	**Values** '*modification date*']	**Uses** Specifies the date the table was last modified
:prol ⋮ ⋮ :eprol :defitem	[FOLD=	{**Y** | N}]	Specifies whether character data or single-byte data in mixed fields in the table is changed to uppercase or left as entered in the table contents
⋮ ⋮	NAME=	*name*	Identifies a table part
:contitem ⋮ ⋮ :row	[TABTYPE=	{**UNSPECIFIED** | MATCHVALID | MATCHINVALID | RANGEMATCH | MESSAGE}]	Defines how the table is used
⋮ ⋮	[TIME=	'*modification time*']	Specifies the time the table was last modified
:vagt	[USAGE=	{**SHARED** | NONSHARED}]	Specifies the default location of the item definition

Syntax	Attributes		
: :	[.]		
:tracbag : :			
:etracbag			
:etble	***:ETBLE*** [.]		

DATE=

'modification date'
Specifies the date the table part was last modified

DATE format is mm/dd/yyyy. Single quotation marks are optional.

DATE=

'modification date'
Specifies the date the table part was last modified

FOLD=

Y Specifies that the contents of a table are folded to uppercase when updated by the user, and that test facility and generation fold the contents as they are used (default value)

N Specifies that the table contents will be used as they were last entered

IMPORT does not fold the table contents. Table definition uses this flag to determine if folding was requested.

NAME=

name Identifies a table part

NAME identifies a table part. It consists of 1 to 7 characters and must meet the following conventions:

- The first character must be alphabetic (A-Z)
- The remaining characters must be alphanumeric (A-Z, 0-9)

- The name cannot contain blanks or have an EZE prefix
- Table names cannot end in 0.

TABTYPE=

UNSPECIFIED
The table is a reference table only. It cannot be used for editing data (as an EDITRTN on the MAPEDITS tag). This type of table can be used only with VisualAge Generator processing statements. (default value)

MATCHVALID
The table can be used for editing data. Data entered by a user must match a value in the first column of the table.

MATCHINVALID
The table can be used for editing data. Data entered by a user must not match any of the values in the first column of the table.

RANGEMATCH
The table can be used for editing data. Data entered by a user must be within a range of values that are contained in the table data.

MESSAGE
The table is used for retrieving user messages when EZEMNO is modified.

TABTYPE defines how the table is used.

TIME=

'modification time'
Specifies the time the table part was last modified

TIME format is hh:mm:ss. Single quotation marks are optional.

USAGE=

SHARED
Indicates that the definition of the item is shared with other references. The definition stored in the library is used. (default)

NONSHARED
Indicates that the definition of the item is unique to this use. The definition stored in the library is not used.

Table generation option specification

The GENOPTS tag specifies the saved options associated with the table. This optional tag further describes the TBLE and ETBLE tag set.

:GENOPTS tag attributes

Syntax	**Attributes**		
:tble . .	*:GENOPTS*		
:genopts . .	**Attributes** [RESIDENT=	**Values** {Y | **N**}]	**Uses** Specifies whether or not a shared table remains in storage after the table use count goes to zero
:etble	[TYPEUSE=	{**SHARED** | SINGLE}]	Specifies whether the table is shared by multiple occurrences of a program
	[.]		

RESIDENT=

Y Specifies that the table remains in storage after the table use count goes to zero

N Specifies that the table is deleted from storage after the table use count goes to zero (default value)

TYPEUSE=

SHARED
Specifies that the table is shared by multiple occurrences of a program or by multiple programs (default value)

SINGLE
Specifies that each occurrence of a program gets its own copy of the table

SINGLE allows writing to a table for temporary storage. Resident tables must be shared.

Table prologue description

The PROL tag begins a text description of a defined table. This optional tag further describes the TBLE and ETBLE tag set. The EPROL tag closes the description, and if you use the PROL tag, the EPROL tag is required.

:PROL tag values

Syntax	**Usage**	
:tble . . .	***:PROL***	
:prol . . .	**Values** [.[*prologue lines*]]	**Uses** Describes the table
:eprol . . .	***:EPROL*** [.]	
:etble		

Only the library size and machine size limit the number of lines that you can specify. Each line can contain up to 60 characters. You can use both uppercase and lowercase in the prologue text. Prologues are saved as entered. When text is longer than 60 characters, it is split into multiple lines during import.

Table column definition

The DEFITEM tag specifies the name and characteristics of the data item defining a column (or part of a column) in a table row. This optional tag further defines the TBLE and ETBLE tag set.

A table row corresponds to a record in a file.

RANGEMATCH and MESSAGE tables require two columns, while tables of other types require one. A table can have more columns than required.

A MESSAGE table contains at least two columns. The first column is defined with numeric data type (NUM only), with a length of 4 and no decimals. The second column is defined with a character or mixed data type and must be no longer than the EZEMSG field. Otherwise, the data will be truncated.

You can repeat the DEFITEM tag multiple times. The order of appearance of DEFITEM tags determines the order of the columns in the table. Specify the BYTES, DECIMALS, DESC, and TYPE attributes for nonshared data items only.

When the column definition (as specified on the DEFITEM tag) matches the table contents (as specified on the ROW tag), the CONTITEM tag is not required. The contents definition is derived from the DEFITEM tag if no CONTITEM tags are specified.

:DEFITEM tag attributes

Syntax	**Attributes**		
:tble . .	***:DEFITEM***		
:defitem . .	**Attributes**	**Values**	**Uses**
	[BYTES=	*field length in bytes*]	Specifies the field length in bytes for the data item based on the data item type
:etble	[DECIMALS=	{**0** | *decimal places*}]	Specifies the number of decimal places
	[DESC=	*'field description'*]	Describes a data item
	[LEVEL=	{**10** | *number*}]	Specifies placement of a data item relative to adjacent data items
	NAME=	{***** | *item name*}	Specifies a data item name
	[TYPE=	{BIN | **CHA** | DBCS | HEX | MIX | NUM | NUMC | PACK | PACF | UNICODE}]	Specifies the data item type

Syntax	Attributes		
	[USAGE= [.]	{SHARED | SHARED}]	Specifies the location of the item definition

BYTES=

field length in bytes
Specifies the field length in bytes for the data item based on the data item type.

3 Default for CHA, MIX, NUM and NUMC

2 Default for BIN, PACK, PACF, and HEX

6 Default for DBCS and UNICODE

BYTES is the number of bytes required to store a data item value internally. The following table lists the valid BYTES values by data type:

Item Type	Byte Values
CHA, MIX	1-254
DBCS, UNICODE	2-254 (even only)
HEX	1-127
NUM, NUMC	1-18
PACK, PACF	1-10
BIN	2, 4, 8 only

Note: BYTES only applies when the item is a nonshared data item or table contents are supplied but the CONTITEM tags are not specified.

DECIMALS=

0 Specifies no decimal places (default value)

decimal places
Specifies the number of decimal places

DECIMALS is the number of positions to the right of an implied decimal point in numeric items. The length specified for the data item must include the places set aside for decimal positions. The maximum number of decimal

positions is 18 or the number of digits defined for the data item, whichever is smallest. The decimal point is not stored with the data.

Note: DECIMALS only applies when the item is a nonshared data item or when table contents are supplied but the CONTITEM tag is not specified.

DESC=

'field description'
Describes a data item

DESC consists of 1 to 30 characters that can be entered in uppercase and lowercase. Mixed data is also permitted.

Note: DESC only applies when the item is a nonshared data item.

LEVEL=

10 Default level number

number
Specifies the relative placement of a data item to adjacent data items in a data structure

LEVEL is a number (3-49, or 77 for single data items in working storage) that is unique to a record, table, or working storage definition. Level numbers can differ for the same data item that is used in several record, table, or working storage definitions.

Data items with the lowest level number in a structure occupy the highest position in the structure. Data items with higher level numbers represent substructures of the previous item in the structure list with a lower level number. The data items with the lowest level number represent the table columns.

The total length of the data items in a substructure must equal the length of the owning data item.

NAME=

***** Specifies a filler data item

item name
Specifies the data item name

NAME consists of 1 to 32 characters and must meet the following conventions:

- The first character must be alphabetic or national (A-Z, $, #, @)
- The remaining characters must be alphanumeric or national characters, hyphens, or underscores (A-Z, 0-9, $, #, @, -, _)
- The name cannot contain blanks or have an EZE prefix.

The name can be a DBCS name up to 15 DBCS characters long with no embedded blanks.

A data item name in a table structure can be specified with an asterisk. This type of data item, called a filler, cannot be referred to by the program; it acts as a space holder.

Note: A filler is always a nonshared data item.

TYPE=

BIN Binary number

CHA Character data (default value)

DBCS Double-byte character data

HEX Hexadecimal data

MIX DBCS data intermingled with single-byte character data

NUM Numeric characters with positive sign in F format

NUMC
Numeric characters with positive sign in C format

PACF Packed decimal characters with positive sign in F format

PACK Packed decimal characters with positive sign in C format

UNICODE
Unicode character data (only valid for Developer on JAVA)

TYPE specifies the internal format or type of data. The data type determines how the item is processed when referred to in processing statements.

Note: TYPE only applies when the item is a nonshared data item or when table contents are supplied but the CONTITEM tag is not specified.

USAGE=

SHARED
Indicates that the definition of the item is shared with other references. The definition stored in the library is used.

NONSHARED
Indicates that the definition of the item is unique to this use. The definition stored in the library is not used.

This USAGE value overrides the value specified on the USAGE attribute of the :TBLE tag. If a USAGE value is not specified, the value specified on the USAGE attribute of the :TBLE tag is the default.

Data content attribute identification

The CONTITEM tag can be repeated multiple times. It specifies the contents structure and identifies the attributes of the actual data contents. These attributes may conflict with those in DEFITEM if the table definition has been changed and the contents have not yet been updated. If the contents structure derived from the DEFITEM tags matches the table contents, the CONTITEM tag is not necessary. The following table lists the data items with the lowest level of the table structure:

Table Structure	Contents Structure
NAME LAST FIRST MIDDLE	NAME
ADDRESS LINE1 LINE2 LINE3	ADDRESS

If CONTITEM tags are not specified but ROW tags are specified, VisualAge Generator derives the contents structure from the DEFITEM tag information (name, level, type, bytes, and decimals). The order of appearance of the

CONTITEM tags determines the order of the columns in the table. The CONTITEM tag is optional and further defines the TBLE and ETBLE tag set.

:CONTITEM tag attributes

Syntax	Attributes		
:tble ⋮ ⋮	*:CONTITEM*		
:contitem ⋮ ⋮	**Attributes** [BYTES=	**Values** *field length in bytes*]	**Uses** Specifies the field length in bytes for the data item based on data item type
:etble	[DECIMALS=	{**0** | *decimal places*}]	Specifies the number of decimal places
	NAME=	{* | *item name*}	Specifies a data item name
	[TYPE=	{BIN | **CHA** | DBCS | HEX | MIX | NUM | NUMC | PACK | PACF | UNICODE}]	Specifies the data item type
	[.]		

BYTES=

field length in bytes
Specifies the field length in bytes for the data item based on the data item type.

3 Default for CHA, MIX, NUM and NUMC

2 Default for BIN, PACK, PACF, and HEX

6 Default for DBCS and UNICODE

BYTES is the number of bytes required to store a data item value internally. The following table lists the valid BYTES values by data type:

Item Type	Byte Values
CHA, MIX	1-254
DBCS, UNICODE	2-254 (even only)
HEX	1-127
NUM, NUMC	1-18
PACK, PACF	1-10
BIN	2, 4, 8 only

DECIMALS=

0 Specifies no decimal places (default value)

decimal places
Specifies the number of decimal places

DECIMALS is the number of positions to the right of an implied decimal point in numeric items. The length specified for the data item must include the places set aside for decimal positions. The maximum number of decimal positions is 18 or the number of digits defined for the data item, whichever is smallest. The decimal point is not stored with the data.

NAME=

***** Specifies a filler data item

item name
Specifies the data item

NAME consists of 1 to 32 characters and must meet the following conventions:

- The first character must be alphabetic or national (A-Z, $, #, @)
- The remaining characters must be alphanumeric or national characters, hyphens, or underscores (A-Z, 0-9, $, #, @, -, _)
- The name cannot contain blanks or have an EZE prefix.

The name can be a DBCS name up to 15 DBCS characters long with no embedded blanks.

A data item name in a table structure can be specified with an asterisk. This type of data item, called a filler, cannot be referred to by the program; it acts as a space holder.

Note: A filler is always a nonshared data item.

TYPE=

BIN Binary number

CHA Character data (default value)

DBCS Double-byte character data

HEX Hexadecimal data

MIX DBCS data intermingled with single-byte character data

NUM Numeric characters with positive sign in F format

NUMC
Numeric characters with positive sign in C format

PACF Packed decimal characters with positive sign in F format

PACK Packed decimal characters with positive sign in C format

UNICODE
Unicode character data (only valid for Developer on JAVA)

TYPE specifies the internal format or type of data. The data type determines how the item is processed when referred to in processing statements.

Row specification

The ROW tag contains the contents of a single table row in character format. This tag is optional, but you can repeat it once for each row of data in the table. You cannot specify the ROW tag unless you specify the DEFITEM or CONTITEM tags. If you do not specify the CONTITEM tag, VisualAge Generator derives the contents structure from the DEFITEM tags. The order of the ROW tags determines the order of the rows in the table.

:ROW tag values

Syntax	**Usage**	
:tble ⋮ ⋮	*:ROW*	
:contitem ⋮ ⋮ **:row**	**Values** [.[*row content*]]	**Uses** Contains the contents of one row in the table

Syntax	Usage
. . . :etble	

Columns of data must be separated by one or more blanks. If the data needs to contain a blank, or starts with a single quotation mark, enclose the column in single quotation marks. When the data needs to contain quotation marks, use double sets of quotation marks. You can use multiple lines to specify a row's contents. To add a line to the specification of a row's contents, place a continuation character in column 72 of the last line currently defined and add the new line immediately below. Any character in column 72 is considered a continuation character. The continuation character causes concatenation of the two lines on import. The two lines concatenate with the character in column 71 being immediately followed by the character in column 1 of the next line. Add lines to a row's contents specification until it is complete.

Templates tracability information

Templates tracability information is defined for a table part as described in "Chapter 16. Templates traceability information structures" on page 213.

Table definition syntax example

```
:tble      name     = xxxxxxx      date    = 'mm/dd/yyyy'
           time      = 'hh:mm:ss'
           tabtype   = xxxxxxxxxxx fold    = x     usage = xxxxxxxxx.
:genopts   typeuse  = xxxxxx       jobname = xxxxxxxx
           fold     = x            lines   = xxx
           resident = x            print   = x
           options  = 'xxxxxxxx.xxxxxxxx'.
:prol.
A prologue line will be split if it is more than 60 characters long.
:eprol.
:defitem   name     = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
           level    = xx           type  = xxx        bytes = xxx
           decimals = xx       desc  = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
           usage    = xxxxxxxxx.
:contitem  name     = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
           type     = xxxx         bytes = xxx         decimals = xx.
:row.'These are the row contents. This can span more than one line and
is split on character boundaries because table data is just a stream of
 characters.  The next row starts with another :row in column 1.'
:row.Thisrowhasnosurroundingquotesbecausetheyarenotnecessaryiftherearen
oblanksorquotesintherowcontents
:row.'This is another row''s contents.  It also spans more than one line
 and is split on character boundaries.  The table contents are terminated
with a :etble in column 1 that is coming up.'
:etble.
```

Note: This example illustrates the syntax of all table definition tags and attributes. Actual definitions do not require or use all tags and attributes shown here.

Chapter 7. Data item structures

This chapter describes the external source format tags and attributes that define a VisualAge Generator data item.

Data item definition

The ITEM tag defines attributes for individual data item parts. The MAPEDITS, MESSAGES, UIPROP, GENEDITS, UIMSGS, NUMEDITS, FLDHELP, and LABEL tags further define aspects of an item and can have attributes. The EITEM tag closes the definition, and if you use the ITEM tag, the EITEM tag is required.

:ITEM tag attributes

Syntax	**Attributes**		
:item ⋮ ⋮	***:ITEM***		
:mapedits ⋮ ⋮ :messages ⋮ ⋮ :uiprop :genedits ⋮ ⋮ :uimsgs ⋮ ⋮ :numedits ⋮ ⋮ :fldhelp ⋮ ⋮ :efldhelp :label	**Attributes** [BYTES= [DATE= [DECIMALS= [DESC= [EVENSQL= NAME= [TIME= [TYPE= [.]	**Values** *field length in bytes*] '*modification date*'] {**0** | *decimal places*}] *'field description'*] {Y | **N**}] *item name* '*modification time*'] {BIN | **CHA** | DBCS | HEX | MIX | NUM | NUMC | PACF | PACK | UNICODE}]	**Uses** Specifies the field length in bytes for the data item based on the data item type Specifies the date the data item was last modified Specifies the number of decimal places Describes the field Specifies whether packed fields of even length or odd length are passed to a relational database Specifies a data item part in the library Specifies the time the data item was last modified Specifies the data item type

Syntax	Attributes		
. . :elabel :euiprop :vagt . . :tracbag . . :etracbag **:eitem**	***:EITEM*** [.]		

BYTES=

field length in bytes
: Specifies the field length in bytes for the data item based on the data item type

3
: Default for CHA, MIX, NUM, and NUMC

2
: Default for BIN, PACK, PACF, and HEX

6
: Default for DBCS and UNICODE

BYTES is the number of bytes required to store a data item value internally. The following table lists the valid BYTES values by data type:

Item Type	Byte Values
CHA, HEX, MIX	1-32,767
DBCS, UNICODE	2-32,766 (even only)
NUM, NUMC	1-18
PACK, PACF	1-10
BIN	2, 4, 8 only

DATE=

'modification date'
: Specifies the date the data item part was last modified

DATE format is mm/dd/yyyy. Single quotation marks are optional.

DECIMALS=

0 Specifies no decimal places (default value)

decimal places
Specifies the number of decimal places

DECIMALS is the number of positions to the right of an implied decimal point in numeric items. The length specified for the data item must include the places set aside for decimal positions. The maximum number of decimal positions is 18 or the number of digits defined for the data item, whichever is smaller. The decimal point is not stored with the data.

DESC=

'field description'
Describes what the data item represents

DESC is a text description of what the data item represents. The text can consist of up to 30 characters, not including the quotation marks, and can be entered in uppercase and lowercase. Mixed data is also permitted.

EVENSQL=

Y Specifies whether packed fields of even length are passed to a relational database

N Specifies whether packed fields of odd length are passed to a relational database (default value)

EVENSQL indicates whether VisualAge Generator passes even- or odd-length packed fields to relational databases. Specify EVENSQL as Y only if you are using an SQL table that has even-length columns defined.

NAME=

name Specifies the data item part

NAME consists of 1 to 32 characters and must meet the following conventions:

- The first character must be alphabetic or national (A-Z, $, #, @)
- The remaining characters must be alphanumeric or national characters, hyphens, or underscores (A-Z, 0-9, $, #, @, -, _)
- The name cannot contain blanks or have an EZE prefix.

The name can be a DBCS name up to 15 DBCS characters long with no embedded blanks.

TIME=

'modification time'
Specifies the time the data item part was last modified

TIME format is hh:mm:ss. Single quotation marks are optional.

TYPE=

BIN Binary number

CHA Character data (default value)

DBCS Double-byte character data

HEX Hexadecimal data

MIX DBCS data intermingled with single-byte character data

NUM Numeric characters with positive sign in F format

NUMC
Numeric characters with positive sign in C format

PACF Packed decimal characters with positive sign in F format

PACK Packed decimal characters with positive sign in C format

UNICODE
Unicode character data (only valid for Developer on JAVA)

TYPE specifies the internal format or type of data. The data type determines how the item is processed when referred to in processing statements.

Map edit characteristic definition

The MAPEDITS tag defines the default map edit characteristics for a data item. This optional tag with its attributes further describes the ITEM and EITEM tag set.

:MAPEDITS tag attributes

Syntax	**Attributes**		
:item ⋮ ⋮	***:MAPEDITS***		
:mapedits ⋮ ⋮	**Attributes** [CURRSYMB=	**Values** {Y | **N**}]	**Uses** Specifies whether the currency symbol is supported
:eitem	[DATEFORM=	{*'date edit mask'* | SYSGREGRN | SYSJULIAN | *number*}]	Specifies the date format editing
	[EDITRTN=	*edit routine*]	Specifies a routine or edit table for special data editing
	[FILLCHAR=	{'N' | *'fill character'*}]	Specifies the character used to fill unused data item positions
	[FLDFOLD=	{Y | **N**}]	Specifies whether data entered in this field is folded
	[HEXEDIT=	{Y | N}]	Specifies whether only hexadecimal digits can be entered in the input field
	[INPUTREQ=	{Y | **N**}]	Specifies whether valid data must be entered
	[JUSTIFY=	{LEF | RIG | N}]	Specifies the position of data when it is shorter than the length of the field
	[MININPUT=	{**N** | *positions*}]	Specifies the minimum number of required characters
	[NUMSEP=	{Y | **N**}]	Specifies whether data can contain numeric separators
	[RANGE=	*lowvalue highvalue*]	Specifies the range of valid numeric values
	[SIGN=	{LEA | TRA | N}]	Specifies a sign in a field as leading, trailing, or none

Syntax	Attributes		
	[SOSI=	{Y | N}]	Specifies whether to check mixed data to see if it will fit into the field after conversion.
	[ZEROEDIT=	{Y | **N**}]	Specifies the format for numeric fields that have zero values
	[.]		

CURRSYMB=

Y Specifies that the currency symbol is supported

N Specifies that the currency symbol is not supported (default value)

CURRSYMB indicates whether the currency symbol is supported in the field. Y is valid only for numeric fields. When defining field length, remember that the currency symbol takes up one position.

DATEFORM=

'date edit mask'
Specifies a valid date edit mask

SYSGREGRN
Specifies system default Gregorian format

SYSJULIAN
Specifies system default Julian format

number
Specifies a predefined date format is used when data is entered or displayed

DATEFORM indicates the date edit format of a map field.

Date edit mask has a maximum length of 10 and consists of DD, MM, YY, DDD, or YYYY in any order separated by a non-numeric single-byte character, except D, M, and Y. If a single quotation mark is used as a separator character it must be specified as two single quotation marks in succession.

When a Gregorian date edit is specified, the data item must follow these conventions:

- Numeric data items must be at least 6 digits for the short format and 8 digits for the long format. Use the short format to refer to dates with a 2-digit year, and use the long format to refer to dates with a 4-digit year.
- Character data items must be at least 8 bytes for the short format and 10 bytes for the long format.

When a Julian date edit is specified, the data item must follow these conventions:

- Numeric data items must be at least 5 digits for the short format and at least 7 digits for the long format. Use the short format to refer to dates with a 2-digit year, and use the long format to refer to dates with a 4-digit year.
- Character data items must be at least 6 bytes for the short format and at least 8 bytes for the long format.

Number consists of any integer from 1 to 17 that represents a predefined date edit mask, as indicated in the following table.

Option Number	Short Date Format	Long Date Format
1	MM/DD/YY	MM/DD/YYYY
2	MM-DD-YY	MM-DD-YYYY
3	MM:YY	MM:YYYY
4	YY/MM/DD	YYYY/MM/DD
5	YY-MM-DD	YYYY-MM-DD
6	YY:MM	YYYY:MM
7	DD/MM/YY	DD/MM/YYYY
8	DD-MM-YY	DD-MM-YYYY
9	DD:MM:YY	DD:MM:YYYY
10	YY-DDD	YYYY-DDD
11	YY:DDD	YYYY:DDD
12	MM.DD.YY	MM.DD.YYYY
13	YY.MM.DD	YYYY.MM.DD
14	DD.MM.YY	DD.MM.YYYY
15	YY.DDD	YYYY.DDD
16	SYSGREGRN	SYSGREGRN
17	SYSJULIAN	SYSJULIAN

Field edits should be defined as follows when you specify a date edit option:

- CURRSYMB=N

- DECIMALS=0
- HEXEDIT=N
- NUMSEP=N
- SIGN=N

The TYPE must be one of the following values when you specify the date edit option:
- BIN
- PACK
- PACF
- NUM
- NUMC
- CHA

EDITRTN=

edit routine
Specifies a routine or edit table for special editing of data in a variable field

EDITRTN indicates the name of a routine or edit table used for special editing of data that a user entered in a variable field. You can specify one of the following:
- The name of one of the following types of editing tables:
  - Match valid table
  - Match invalid table
  - Range match valid table.
- The name of one of the following special function word subroutines:
  - Modulus 10 check digit routine (EZEC10)
  - Modulus 11 check digit routine (EZEC11).
- The name of a function used as an edit routine.

FILLCHAR=

'N' Specifies a null fill character

'fill character'
Specifies the character used to fill unused data item positions

FILLCHAR indicates the character used to fill unused field positions on output to the terminal or printer. The default fill character for CHA, MIX, and DBCS fields is a null. The default for HEX fields is a 0 and for numeric data (NUM, NUMC, PACK, PACF, or BIN) is a blank.

FLDFOLD=

Y	Specifies that data entered in this field is folded
N	Specifies that data entered in this field is not folded (default value)

FLDFOLD specifies whether lowercase alphabetic characters that the user enters are to be folded (converted) to uppercase.

Note: FLDFOLD is not allowed for DBCS fields.

HEXEDIT=

Y	Specifies that only hexadecimal digits can be entered (default for HEX fields)
N	Specifies the map field is not checked for hexadecimal characters (default for all fields except HEX fields)

HEXEDIT specifies whether the input field is checked for hexadecimal digits. The data type of the variable field must be CHA or HEX for Y to be specified.

For HEX fields, the default is Y. Otherwise, the default is N.

INPUTREQ=

Y	Specifies that data must be entered in a map field
N	Specifies that input is not required in the field (default value)

INPUTREQ indicates whether data must be entered in a map field. When a field contains data other than blanks for character type or zeros for numeric types, the field is considered to have input. Blanks in a character field and zero in a numeric field do not satisfy the input required edit check. Even if blanks for character fields and zeros for numeric fields are valid values, INPUTREQ will return an error message unless the user types data into the field.

JUSTIFY=

LEF	Specifies left justification (default for character data)
RIG	Specifies right justification (default for numeric data)
N	Specifies no justification

JUSTIFY specifies the position of data in a variable field when the data is shorter than the length of the field. If JUSTIFY is not specified, character data is left justified and numeric data is right justified. Right justification is required for numeric data with decimal positions or sign specified. JUSTIFY=N is not valid for numeric fields.

MININPUT=

N	Specifies no minimum number of characters required (default value)
positions	Specifies the minimum number of characters that must be entered in a valid variable field

MININPUT specifies the minimum number of characters that must be entered in a variable field if any data is entered.

NUMSEP=

Y	Specifies that data can contain numeric separators
N	Specifies that data cannot contain numeric separators (default value)

NUMSEP specifies whether data in a field can contain the previously defined numeric separator. The default numeric separator is a comma. Your system administrator can change the default character using the customization procedures for language-dependent options.

If YES is specified, numeric separators are allowed in the field on input. After input editing, the numeric separators are removed before the field is placed in internal storage. On output, numeric separators are inserted between every three significant digits. Every fourth position to the left of the decimal point will be a separator. You can only specify NUMSEP=Y for numeric fields.

When defining field length, remember that each numeric separator takes up one position. If NUMSEP=N is specified, validation is done to ensure that the user did not enter a numeric separator in the field.

RANGE=

lowvalue
Specifies the smallest numeric value for a field

highvalue
Specifies the largest numeric value for a field

RANGE specifies the range of valid numeric values for a field. Lowvalue is the smallest numeric value that can be entered in a specified field. Highvalue is the largest numeric value that can be entered in a specified field. The high and low values must have the same length, number of decimal positions, and sign as defined for the field. Both low and high values must be specified if this attribute is used. The values are separated by a space and cannot be longer than the length of the field being defined. RANGE can be specified only for numeric fields.

SIGN=

LEA	Specifies a leading sign (default value for numeric fields)
TRA	Specifies a trailing sign
N	Specifies no sign (default value for character fields)

SIGN specifies whether a sign is displayed in a field and whether it is a leading or a trailing sign. You can specify signs for numeric fields only. When defining field length, remember that the sign takes up one position.

SOSI=

Y	Specifies that mixed data entered in the map field on an ASCII device is to be checked to see if it will fit into the field after conversion into the EBCDIC SO/SI format for mixed data (default value for mixed data items)
N	Specifies that mixed data entered in the map field on an ASCII device is not to be checked to see if it will fit into the field after conversion into the EBCDIC SO/SI format for mixed data (default value for all item types except mixed data items)

ZEROEDIT=

Y	Specifies the display format for numeric fields containing zero values
N	Specifies that no editing is done on numeric fields containing zero values (default value)

ZEROEDIT specifies the display format for numeric fields containing zero values. The following table gives a list of what a numeric field will contain when ZEROEDIT is specified with either Y or N. The sample field has been defined as right justified with a length of 11. A "b" represents a blank fill character.

DEC (2)	CURRSYMB / NUMSEP	FILL	ZEROEDIT= N	ZEROEDIT= Y
N	N	N	nulls	0
N	N	b	blanks	bbbbbbbbbb0
N	N	0	00000000000	00000000000
N	N	*	**********	**********0
Y	N	N	nulls	0.00
Y	N	b	blanks	bbbbbbb0.00
Y	N	0	00000000000	00000000.00
Y	N	*	**********	*******0.00
Y	Y	N	nulls	$0.00
Y	Y	b	blanks	bbbbbb$0.00
Y	Y	0	00000000000	$000,000.00
Y	Y	*	**********	******$0.00

Data item message definition

The MESSAGES tag specifies the data item messages from a user-defined message table. This optional tag with its attributes further describes the ITEM and EITEM tag set.

Message numbers must be in the range of -9999 to 9999.

:MESSAGES tag attributes

Syntax	Attributes		
:item ⋮ **:messages** ⋮ :eitem	**:MESSAGES** **Attributes** [EDITMSG= [INVALMSG= [MININMSG= [RANGEMSG= [REQMSG= [.]	 **Values** *message number*] *message number*] *message number*] *message number*] *message number*]	 **Uses** Specifies the message number of the edit routine error message Specifies the message number of the data type error message Specifies the message number of the minimum input error message Specifies the message number of the value error message Specifies the message number of the input required error message

EDITMSG=

message number
Specifies the number of the message displayed when the data fails a modulus check or table edit check

If you do not specify this attribute, the default error message for a modulus check is "Modulus check error—reenter", and the default error message for a table edit check is "Table edit validity error—reenter".

INVALMSG=

message number
Specifies the message number of the message displayed when the data entered is incompatible with the variable field data type

If you do not specify this attribute, the default error message for invalid data type is "Data type error in input—reenter".

MININMSG=

message number
Specifies the number of the message displayed when the minimum input edit check fails

If you do not specify this attribute, the default error message for this error is "Input minimum length error—reenter".

RANGEMSG=

message number
Specifies the number of the message displayed when the minimum or maximum value check fails

If you do not specify this attribute, the default error message is "Input not within defined range—reenter".

REQMSG=

message number
Specifies the number of the message displayed when the input required edit check fails

If you do not specify this attribute, the default error message for this error is "No input received for required field—reenter".

User interface properties definition

The UIPROP tag specifies property information about an item. This optional tag further defines the ITEM tag. The optional tags GENEDITS, UIMSGS, NUMEDITS, FLDHELP, and LABEL provide additional user interface information about the data item. The EUIPROP tag closes the definition and is required.

:UIPROP tag attributes

There are no attributes for this tag set.

General edit characteristic definition

The GENEDITS tag defines a list of general editing characteristics. This optional tag further defines the UIPROP and EUIPROP tag set.

:GENEDITS tag attributes

Syntax	Attributes		
:item ⋮	***:GENEDITS***		
:uiprop ⋮	**Attributes**	**Values**	**Uses**
	[EDITFUNC=	*function name*]	Specifies the function used for special data editing
:genedits	[EDITTBLE=	*table name*]	Specifies the table used for special data editing
⋮	[EDITTYPE=	{BOOLEAN | DATE | TIME | **NONE**}]	Specifies the type of editing to be done on the data item
:euiprop	[FILLCHAR=	*'fill character'*]	Specifies the character used to fill unused field positions
⋮	[FLDFOLD=	{Y | **N**}]	Specifies whether data entered into the field is folded
:eitem	[INPUTREQ=	{Y | **N**}]	Specifies whether valid data must be entered
	[MININPUT=	{**N** | *positions*}]	Specifies the minimum number of required characters
	[RUNATWEB=	{Y | **N**}]	Specifies whether edits are to be performed at the WEB site or at the VG Server
	[SOSI=	{Y | N}]	Specifies whether to check mixed data to see if it will fit into the field after conversion.
	[.]		

EDITFUNC=

function name
Specifies the name of a function used for special editing of data in a variable field

EDITFUNC indicates the name of a routine to be used for special editing of data that the user enters in a variable field. You can specify one of the following:

- The name of one of the following special funtion word subroutines:
  - Modulus 10 check digit routine (EZEC10)
  - Modulus 11 check digit routine (EZEC11)
- The name of an edit routine.

EDITTBLE=

table name
Specifies the name of a table used for special editing of data in a variable field

EDITTYPE=

BOOLEAN
Specifies to the UI record that the data in the item is in the form of a Boolean value.

DATE Specifies to the UI record that the data in the item is in the form of a Date value.

TIME Specifies to the UI record that the data in the item is in the form of a Time value.

NONE
Specifies to the UI record that the data in the item is in no particular form. (default value)

FILLCHAR=

'fill character'
Specifies the character used to fill unused field positions

FILLCHAR indicates the character used to fill unused field positions on output.

FLDFOLD

Y Specifies that data is folded

N Specifies that data is not folded (default value)

FLDFOLD specifies whether lowercase alphabetic characters entered by the user are to be folded (converted) to uppercase.

INPUTREQ=

Y Specifies that data must be entered in the field

N Specifies that input is not required in the field (default value)

INPUTREQ indicates whether data must be entered in a field. When a field contains data other than blanks for character type or zeros for numeric types, the field is considered to have input. Blanks in a character field and zero in a numeric field do not satisfy the input required edit check. Even if blanks and zeros are valid values, INPUTREQ returns an error message unless the user types data into the field.

MININPUT=

N Specifies no minimum number of characters required (default value)

positions
Specifies the minimum number of characters that must be entered in a valid variable field

MININPUT specifies the minimum number of characters that must be entered in a variable field if any data is entered.

RUNATWEB=

Y Specifies that the edits are to be performed at the WEB Server site

N Specifies that the edits are to be performed by the VG Server program

SOSI=

Y Specifies that mixed data entered in the field on an ASCII device is to be checked to see if it will fit into the field after conversion into the EBCDIC SO/SI format for mixed data (default for mixed type fields)

N Specifies that mixed data entered in the field on an ASCII device is not to be checked to see if it will fit into the field after conversion into the EBCDIC SO/SI format for mixed data (default for all fields except mixed type fields)

Message key definition

The UIMSGS tag specifies property information about message keys for a data item. This optional tag further defines the UIPROP and EUIPROP tag set.

:UIMSGS tag attributes

Syntax	**Attributes**		
:item : :	*:UIMSGS*		
:uiprop : :	**Attributes**	**Values**	**Uses**
	[FUNCKEY=	*'edit function message key'*]	Specifies the message key to be used when an edit function is used to determine an error condition exists
:uimsgs	[MININKEY=	*'minimum input message key'*]	Specifies the message key to be used when fewer than the minimum characters are input

Syntax	Attributes		
. . .	[RANGEKEY=	*'numeric range message key'*]	Specifies the message key to be used when the user enters data that is out of range for this field
:euiprop	[REQKEY=	*'input required message key'*]	Specifies the message key to be used when the user does not enter data into a field that must contain data
. . .	[TBLEKEY=	*'edit table message key'*]	Specifies the message key to be used when an edit table is used to determine an error condition exists
	[TYPEKEY=	*'data type message key'*]	Specifies the message key to be used when the type of data entered by the user is not valid for the field
:eitem	[.]		

FUNCKEY=

'edit function message key'
Specifies the message key to be used when an edit function is used to determine an error condition exists

MININKEY=

'minimum input message key'
Specifies the message key to be used when fewer than the minimum characters are input

RANGEKEY=

'numeric range message key'
Specifies the message key to be used when the user enters data that is out of range for this field

REQKEY=

'input required message key'
Specifies the message key to be used when the user does not enter data into a field that must contain data

TBLEKEY =

'edit table message key'
Specifies the message key to be used when an edit table is used to determine an error condition exists

TYPEKEY=

'data type message key'
Specifies the message key to be used when the type of data entered by the user is not valid for the field

Numeric edit characteristic definition

The NUMEDITS tag defines a list of numeric editing characteristics. This optional tag further defines the UIPROP and EUIPROP tag set. It can only be specified for numeric data items.

:NUMEDITS tag attributes

Syntax	**Attributes**		
:item . .	*:NUMEDITS*		
:uiprop	**Attributes**	**Values**	**Uses**
.	CURRENCY=	{Y | **N**}]	Specifies whether a default currency symbol is to be used
	[CURRSYMB=	'*symbol*']	Specifies a 1- to 3-character currency symbol
:numedits			
.	[NUMSEP=	{Y | **N**}]	Specifies whether data can contain numeric separators
	[RANGE=	lowvalue highvalue]	Specifies the range of valid numeric values
:uiprop			

Syntax	Attributes		
:	[SIGN=	{LEA | TRA | N}]	Specifies a sign in the field as leading, trailing, or none
	[ZEROEDIT=	{Y | **N**}]	Specifies the format for fields that have zero values
:eitem	[.]		

CURRENCY=

Y Specifies that the currency symbol is to be used

N Specifies that no currency symbol is to be used (default value)

CURRENCY indicates whether the currency symbol is supported in the field. Y is valid only for numeric fields. When defining field length, there must be enough room for each character specified on CURRSYMB.

CURRSYMB=

'symbol'
Specifies a 1- to 3-character currency symbol to be used if CURRENCY is Y

NUMSEP=

Y Specifies that a numeric separator is to be used with the field

N Specifies that a numeric separator is not to be used with the field (default value)

NUMSEP specifies whether data in a field can contain the previously defined numeric separator. The default numeric separator is a comma. Your system administrator can change the default character using the customization procedures for language-dependent options.

If YES is specified, numeric separators are allowed in the field on input. After input editing, the numeric separators are removed before the field is placed in internal storage. On output, numeric separators are inserted between every three significant digits. Every fourth position to the left of the decimal point will be a separator. You can only specify NUMSEP=Y for numeric fields. When defining field length, remember that each numeric separator takes up

one position. If NUMSEP=N is specified, validation is done to ensure that the user did not enter a numeric separator in the field.

RANGE=

lowvalue
Specifies the smallest numeric value for a field

highvalue
Specifies the largest numeric value for a field

RANGE specifies the range of valid numeric values for a field. Lowvalue is the smallest numeric value that can be entered in a specified field. Highvalue is the largest numeric value that can be entered in a specified field. The high and low values must have the same length, number of decimal positions, and sign as defined for the field. Both low and high values must be specified if this attribute is used. The values are separated by a space and cannot be longer than the length of the field being defined. RANGE can be specified only for numeric fields.

SIGN=

LEA	Specifies a leading sign
TRA	Specifies a trailing sign
N	Specifies no sign (default value)

SIGN specifies whether a sign is displayed in a field and whether it is a leading or a trailing sign. You can specify signs for numeric fields only. When defining field length, remember the sign takes up one position.

ZEROEDIT=

Y	Specifies the display format for numeric fields containing 0 values
N	Specifies that no editing is done on 0 numeric fields (default value)

ZEROEDIT specifies the display format for numeric fields that have 0 values.

Field help definition

The FLDHELP tag begins the set of text lines that are to be displayed as help for this field. This optional tag further defines the UIPROP and EUIPROP tag set. The EFLDHELP tag closes the definition, and if you use the FLDHELP tag, the EFLDHELP tag is required.

The FLDHELP tag can appear only once for each item within a record definition.

:FLDHELP tag values

Syntax	Usage	
:item ⋮ ⋮	***:FLDHELP***	
:uiprop ⋮ ⋮	**Values** [.[*field help text*]]	**Uses** Specifies the help text for a data item
:fldhelp ⋮ ⋮	***:EFLDHELP*** [.]	
:efldhelp :eitem	[.]	

Only the library size and machine size limit the number of lines that you can specify. Each line can contain up to 2000 bytes. You can use both uppercase and lowercase characters. The help text must be preceded by a period (.). Text is saved as entered.

Label definition

The LABEL tag begins the text lines that are to be displayed as the label text for this field. This optional tag further defines the UIPROP and EUIPROP tag set. The ELABEL tag closes the definition, and if you use the LABEL tag, the ELABEL tag is required.

The LABEL tag can appear only once for an item within a record definition.

:LABEL tag values

Syntax	Usage	
:item ⋮ ⋮	***:LABEL***	

Syntax	Usage	
:label : :	**Values** [.[*label text*]]	**Uses** Specifies the label for an item
:elabel : :	***:ELABEL*** [.]	
:eitem	[.]	

Each label must be contained on a single line. A single line cannot exceed 2000 bytes. You can use both uppercase and lowercase characters. The label text must be preceded by a period (.). Text is saved as entered.

Templates tracability information

Templates tracability information is defined for a data item part as described in "Chapter 16. Templates traceability information structures" on page 213.

Data item definition syntax example

```
:item    name     = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
           assocto = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
           date  = 'mm/dd/yyyy'  time     = 'hh:mm:ss' type     = xxxx
           bytes = xxxxx       decimals = xx       evensql  = x
           desc     = 'A 30 character description'.
:mapedits  fldfold  = x
           range    = xxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxx
           mininput = xx
           fillchar = 'x'
           editrtn  = xxxxxxxxxxxxxxxxxx
           dateform = 'xxxxxxxxxx'
           currsymb = x   sosi     = x   sign     = xxx numsep   = x
           inputreq = x   justify  = xxx hexedit  = x   zeroedit = x.
:messages  invalmsg = xxxxx
           mininmsg = xxxxx
           reqmsg   = xxxxx
           editmsg  = xxxxx
           rangemsg = xxxxx.
:uiprop.
:genedits  edittype  = xxxx
           editfunc  = xxxxxxxxxxxxxxxxx
           edittble  = xxxxxxx
           fillchar  = 'x'
           fldfold   = x
           inputreq  = x
           mininput  = xxxxx
           runatweb  = x
           sosi      = x.
:uimsgs    tblekey   = 'xxxxxxxxxx'
           funckey   = 'xxxxxxxxxx'
           mininkey  = 'xxxxxxxxxx'
           rangekey  = 'xxxxxxxxxx'
           reqkey    = 'xxxxxxxxxx'
           typekey   = 'xxxxxxxxxx'.
:numedits  currency  = x    sign = xxx     numsep = x
           currsymb  = 'xxx'
           range     = xxxxxxxxxxxxxxxxxx    xxxxxxxxxxxxxxxxxx
           zeroedit  = x.
:fldhelp.
Field help text goes here
:efldhelp.
:label.
Field label text goes here
:elabel.
:euiprop.
:eitem.
```

Note: This example illustrates the syntax of all data item definition tags and attributes. Actual definitions do not require or use all tags and attributes shown here.

Chapter 8. Program specification block structures

This chapter describes the external source format tags and attributes that define a VisualAge Generator program specification block (PSB).

PSB definition

The PSB tag and its attributes define PSB parts.

The PSB tag specifies a set of DL/I database structures that a program can access. The PSB part contains a subset of the information in a DL/I PSB. The PSB structure also identifies the PCBs used for terminal, printer, and message queue support in the IMS/VS and IMS BMP environments.

The PCB and SENSEG tags further define aspects of a PSB part and can have attributes. The EPSB tag closes the definition, and if you use the PSB tag, the EPSB tag is required.

:PSB tag attributes

Syntax	Attributes		
:psb : :	***:PSB***		
:pcb : :	**Attributes** [DATE=	**Values** *'modification date'*]	**Uses** Specifies the date the PSB member was last modified
:senseg : :	NAME= [TIME=	*name* *'modification time'*]	Specifies a PSB member Specifies the time the PSB member was last modified
:vagt : :	[.]		
:tracbag : :			
:etracbag **:epsb**	***:EPSB*** [.]		

DATE=

'modification date'
Specifies the date the PSB part was last modified

DATE format is mm/dd/yyyy. Single quotation marks are optional.

NAME=

name Specifies the PSB part

NAME specifies a PSB part. It consists of 1 to 8 characters and must meet the following conventions:

- The first character must be alphabetic or national (A-Z, $, #, @)
- The remaining characters must be alphanumeric or national (A-Z, 0-9, $, #, @)
- The name cannot contain blanks or have an EZE prefix.

TIME=

'modification time'
Specifies the time the PSB part was last modified

TIME format is hh:mm:ss. Single quotation marks are optional.

Program communication block (PCB) specification

The PCB tag specifies a program communication block entry in the DL/I program specification block. This tag further describes the PSB and EPSB tag set.

The PCB tag can be repeated once for each program communication block in the DL/I program specification block.

:PCB tag attributes

Syntax	Attributes		
:psb . . .	***:PCB***		

Syntax	Attributes		
:pcb	**Attributes**	**Values**	**Uses**
. . .	[DBNAME=	*database name*]	Specifies a database used with the PSB
:senseg	[TYPE=	{TP | **DB** | GSAM}]	Specifies the type of PCB
. . .	[.]		
:epsb			

DBNAME=

database name
Specifies a database used with the PSB

DBNAME is required for DB and GSAM PCBs and cannot be specified for TP PCBs.

DBNAME consists of 1 to 8 characters and must meet the following conventions:

- The first character must be alphabetic or national (A-Z, $, #, @)
- The remaining characters must be alphanumeric or national (A-Z, 0-9, $, #, @)
- The database name cannot contain blanks or have an EZE prefix.

TYPE=

TP Specifies a teleprocessing PCB

DB Specifies a database PCB (default value)

GSAM
Specifies a GSAM PCB

TYPE specifies the type of PCBs.

Specify PCBs in the following order:
1. TP
2. DB
3. GSAM.

Segment sensitivity specification

The SENSEG tag specifies segment sensitivity for a PCB entry in the DL/I PSB. Its attributes further describe the PSB and EPSB tag set. This tag is required for TYPE DB unless DBNAME is ELAWORK or ELAMSG. ELAWORK and ELAMSG represent work and message databases required for running the program with VisualAge Generator Server for MVS, VSE, and VM. SENSEG is not allowed for teleprocessing (TP) and GSAM PCBs.

The SENSEG tag can be repeated for each sensitive segment in a DB PCB. The order of the SENSEG tags determines the order of the sensitive segments in the PCB.

:SENSEG tag attributes

Syntax	**Attributes**		
:psb . .	***:SENSEG***		
:pcb . .	**Attributes**	**Values**	**Uses**
	[IKEY=	*secondary index field name*]	Specifies the secondary index key field
:senseg	[PARENT=	{0 | *parent name*}]	Specifies the segment's parent segment
. . .	SEGMENT=	*segment name*	Specifies a segment in the database
:epsb	[.]		

IKEY=

secondary index field name
Specifies the secondary index key field

IKEY must be the name of an item defined in the segment or in working storage. IKEY must also be the same name defined in the NAME operand in the XDFLD statement that defines the secondary index field in the DL/I database description (DBD). You can specify IKEY for the root segment only.

IKEY consists of 1 to 8 characters and must meet the following conventions:

- The first character must be alphabetic or national (A-Z, $, #, @)
- The remaining characters must be alphanumeric or national (A-Z, 0-9, $, #, @)

- The secondary index field name cannot contain blanks or have an EZE prefix.

PARENT=

0 Specifies 0 as the parent (only valid for the root segment)

parent name
Specifies the segment's parent segment

PARENT must be the name defined in the PARENT operand in the SENSEG statement in the DL/I PSB. PARENT is required for all but the root segment.

PARENT consists of 1 to 8 characters and must meet the following conventions:

- The first character must be alphabetic or national (A-Z, $, #, @)
- The remaining characters must be alphanumeric or national (A-Z, 0-9, $, #, @)
- The parent name cannot contain blanks or have an EZE prefix.

SEGMENT=

segment name
Specifies a segment in the database

SENSEG must be the same as defined in the NAME operand in the SENSEG statement in the DL/I PSB.

SEGMENT consists of 1 to 8 characters and must meet the following conventions:

- The first character must be alphabetic or national (A-Z, $, #, @)
- The remaining characters must be alphanumeric or national (A-Z, 0-9, $, #, @)
- The segment name cannot contain blanks or have an EZE prefix.

Templates tracability information

Templates tracability information is defined for a PSB part as described in "Chapter 16. Templates traceability information structures" on page 213.

PSB definition syntax example

```
:psb       name    = xxxxxxxx    date   = 'mm/dd/yyyy'  time = 'hh:mm:ss'
                 assocto = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'.
:pcb       dbname  = xxxxxxxx    type   = xxxx.
:senseg    segment = xxxxxxxx    parent = 0         ikey = xxxxxxxx.
:pcb       dbname  = xxxxxxxx    type   = xxxx.
:senseg    segment = xxxxxxxx    parent = 0         ikey = xxxxxxxx.
:senseg    segment = xxxxxxxx    parent = xxxxxxxx.
:pcb       dbname  = ELAWORK     type   = xxxx.
:pcb       dbname  = ELAMSG      type   = xxxx.
:pcb       dbname  = xxxxxxxx    type   = xxxx.
:epsb.
```

Note: This example illustrates the syntax of all PSB definition tags and attributes. Actual definitions do not require or use all tags and attributes shown here.

Chapter 9. Map structures

This chapter describes the external source format tags and attributes that define a VisualAge Generator map.

Map definition

The MAP tag defines attributes for individual map parts. The PRESENT, CFIELD, CATTR, VFIELD, MAPEDITS, MESSAGES, and VATTR tags further define aspects of a map and can have attributes. The EMAP tag closes the definition.

Naming fields on a map part does not define the associated data item parts or create the data item parts for fields that are defined in external source format when the map is edited with map definition.

The CFIELD and VFIELD tags describe constant and variable fields on the map. You must describe the fields in the order that they appear on the map (top to bottom, left to right). Therefore, the order of CFIELD and VFIELD tags depends on the constant and variable fields defined on the map.

:MAP tag attributes

Syntax	**Attributes**		
:map ⋮ ⋮	***:MAP***		
:present ⋮ ⋮ :cfield	**Attributes** [BYPKEY=	**Values** *number list*]	**Uses** Specifies up to five keys that allow the user to bypass map edits and map edit groups
⋮ ⋮	[DATE=	'*modification date*']	Specifies the date the map was last modified
:cattr ⋮ ⋮	DEVICES=	*device names*	Specifies the device or devices for which the map is defined
:ecfield	GRPNAME=	*name*	Identifies a group of maps used with a program
⋮ ⋮	[HELPKEY=	*number*]	Specifies the key that displays help

Syntax	Attributes		
:vfield	[HELPMAP=	*map name*]	Specifies the name of a user-defined help map
:	MAPNAME=	*name*	Identifies the map within a map group
:mapedits	[MAPSIZE=	*lines columns*]	Specifies the number of lines and columns for the map
:	[SOSIPOS=	{**Y** | N}]	Specifies whether SO and SI delimiters take a position (this value is valid for DBCS or mixed data)
:messages			
:	[STARTPOS=	{*line column* | **1 1** | NEXT SAME}]	Specifies the starting position of the map
:vattr	[TIME=	'*modification time*']	Specifies the time the map was last modified
:	[.]		
:evfield	***:EMAP*** [.]		
:			
:vagt			
:			
:tracbag			
:			
:etracbag			
:emap			

BYPKEY=

number list
Specifies up to five keys that allow the user to bypass map edits and map edit groups

Specify BYPKEY keys as integers from 1 to 24. Separate multiple keys with blanks. No default bypass edit keys are designated.

Note: Specifying the bypass edit keys for a map overrides the program specification bypass keys for that map. The help key cannot be the same as any of the bypass edit keys.

DATE=

'modification date'
Specifies the date the map part was last modified

DATE format is mm/dd/yyyy. Single quotation marks are optional.

DEVICES=

device names
Specifies the device or devices for which the map is defined

You must specify at least one device. If you specify multiple devices, you must separate them with a blank. The devices you specify must be compatible with each other.

Devices are considered compatible when they are of the same type (display, printer, DBCS display, or DBCS printer), and the map being defined fits within the row and column limitations of the smallest physical device.

The following table lists the valid devices and their sizes.

Supported IBM Devices	Rows	Columns
3643-2	6	40
3277-1	12	40
3643-4	16	64
3278-1, 3278-1B, ANY-1D	12	80
3278-2, 3278-2B, ANY-2D	24	80
5550D (DBCS display device)	24	80
3278-3, 3278-3B, ANY-3D	32	80
3278-4, 3278-4B, ANY-4D	43	80
3278-5, 3278-5B, ANY-5D	27	132
ANY-D (3290 Configured as 62x160)	255	160
3767 PRINT-B PRINTER (physical size 66x132)	255	132
5550P (DBCS printer with size 66x158)	255	158

GRPNAME=

name Identifies a group of maps used with a program

GRPNAME identifies a group of maps used with one program. It consists of 1 to 6 characters and must meet the following conventions:

- The first character must be alphabetic (A-Z)
- The remaining characters are alphanumeric (A-Z, 0-9)
- The name cannot contain blanks or have an EZE prefix.

Map group names cannot contain the characters $, #, and @.

HELPKEY=

number
Specifies the key to use for requesting help from program maps

Specify HELPKEY as an integer ranging from 1 to 24. If you do not specify this attribute, the help key defined for the program is used. You can only specify a help key if a help map name is defined.

Note: The help key cannot be the same as any of the bypass edit keys.

HELPMAP=

map name
Specifies the name of a user-defined help map

HELPMAP must be in the help map group or the main map group specified for the program. The help map cannot have variable fields, and it cannot be a floating map. If the help map is a partial map, it replaces the full screen. The help map must be for a display device and not a printer.

MAPNAME=

name Identifies the map within a map group

MAPNAME consists of 1 to 8 characters and must meet the following conventions:

- The first character must be alphabetic or national (A-Z, $, #, and @).
- The remaining characters must be alphanumeric or national (A-Z, 0-9, $, #, and @).
- The name cannot contain blanks or have an EZE prefix.

MAPSIZE=

lines Specifies the number of lines

columns
Specifies the number of columns

In most cases, MAPSIZE defaults to the smallest of the devices specified for the map. Printer maps default to a size of 66 lines but can be made larger. You can specify the size as less than the default value to define a partial map. Separate the lines value and the column value with a blank.

SOSIPOS=

Y Specifies that SO/SI positions are represented by blanks in the printed output (default value)

N Specifies that SO/SI positions are not represented by blanks in the printed output

SOSIPOS specifies whether shift-out (SO) and shift-in (SI) delimiters take a position when printing mixed data from a program. When directing printer output containing DBCS or mixed data to a system printer, specify SOSIPOS=N if you do not want SO and SI to take a position.

STARTPOS=

line Specifies the line coordinate where the map starts

column Specifies the column coordinate where the map starts

1 Specifies line one (default value)

1 Specifies column one (default value)

NEXT Specifies the next available line within the floating area specified on the AREA tag (in map group specification)

SAME Specifies the next available column within the floating area specified on the AREA tag (in map group specification)

STARTPOS specifies the map's starting position on a screen using a line and column coordinate. Partial maps (maps smaller than the screen size) can share the same screen display if their position allows all of them to be vertically contained within the screen area. Separate the line value and the column value with a blank space. Floating maps are defined by specifying NEXT and SAME.

TIME=

'modification time'
Specifies the time the map part was last modified

TIME format is hh:mm:ss. Single quotation marks are optional.

Presentation information definition

The PRESENT tag defines the presentation information used for the map. This optional tag with its attributes further describes the MAP and EMAP tag set.

:PRESENT tag attributes

Syntax	**Attributes**		
:map : :	*:PRESENT*		
:present : :	**Attributes**	**Values**	**Uses**
	[DEFFOLD=	{Y | **N**}]	Specifies whether character data is folded during online map definition processing
:emap	[TABPOS=	*number list*]	Specifies tab positions used during map definition
	[VARFOLD=	{**Y** | N}]	Specifies whether all character data and all single-byte data in mixed fields is folded at execution time
	[.]		

DEFFOLD=

Y Specifies that character data is folded during map presentation processing

N Specifies that character data is not folded during map presentation processing (default value)

DEFFOLD indicates whether characters entered in the map presentation display of VisualAge Generator map definition are folded. Folding affects online definition processing only. See FLDFOLD on page 174 and VARFOLD for information on how to affect execution processing.

TABPOS=

number list
Specifies tab positions used during map definition

TABPOS allows you to specify ten tab positions; import sorts them into ascending order. You must separate tab positions with blanks. The default is one tab of 1.

VARFOLD=

Y Specifies that all character data and all single-byte data in mixed fields on the map are folded to uppercase (default value)

N Specifies that all character data and all single-byte data in mixed fields remains as entered (uppercase and lowercase) or are folded on a field-to-field basis as specified on the MAPEDITS tag for each VFIELD.

VARFOLD specifies whether all character and all single-byte data entered on a map during execution is folded to uppercase. Folding does not affect numeric or DBCS data.

Constant field definition

The CFIELD tag specifies information about a constant field on the map. The optional tag CATTR provides additional information about a constant field. The ECFIELD tag closes the definition and is required for each CFIELD.

:CFIELD tag attributes

Syntax	**Attributes**		
:map ⋮ ⋮	***:CFIELD***		
:cfield ⋮ ⋮	**Attributes** BYTES=	**Values** *field length in bytes*	**Uses** Specifies the number of positions the constant field occupies
:cattr	COLUMN=	*column number*	Specifies the column of the byte immediately preceding the constant field
⋮ ⋮	ROW=	*row number*	Specifies the row of the byte immediately preceding the constant field
:ecfield ⋮ ⋮	[TYPE= [**.**[*field text*]]	{**CHA** | DBCS | MIX}]	Specifies the type of data Specifies the value of the constant field
:emap	***:ECFIELD*** [.]		

BYTES=

field length in bytes
Specifies the number of positions the constant field occupies

COLUMN=

column number
Specifies the column of the byte immediately preceding the constant field

ROW=

row number
Specifies the row of the byte immediately preceding the constant field

TYPE=

CHA Specifies any character data (default value)

DBCS Specifies double-byte character set, ideographic character data that requires two positions for each character

MIX Specifies DBCS and single-byte data in the same field

field text

field text
Specifies the value of the constant field

Field text begins in the next available byte immediately following the period. The period is required if field text is supplied. If a line of text in the external source format file is longer than 71 characters, place a continuation character in column 72. Any character in column 72 is a continuation character. The continuation character causes concatenation of the two lines on import.

Constant field attribute definition

The CATTR tag defines attributes for the constant described in the CFIELD tag.

:CATTR tag attributes

Syntax	**Attributes**		
:map ⋮ ⋮	***:CATTR***		
:cfield ⋮ ⋮	**Attributes**	**Values**	**Uses**
	[COLOR=	{**MONO** | BLUE | RED | PINK | GREEN | TURQUOISE | YELLOW | WHITE}]	Specifies the color of the text in the field
:cattr	[CURSOR=	{Y | N}]	Specifies if the cursor is positioned on this field
⋮ ⋮	[DATA=	{**ALPHA** | NUMERIC}]	Specifies the field data type
:ecfield	[DETECT=	{Y | **N**}]	Specifies whether the field is light pen detectable

Syntax	Attributes		
:	[ENTER=	{Y|**N**}]	Specifies if data is required to be entered into the field
:emap	[FILL=	{Y|**N**}]	Specifies whether an error occurs when a user does not enter enough characters to fill the field
	[HILITE=	{**NOHILITE**| BLINK| RVIDEO| USCORE}]	Specifies the highlighting characteristics of the constant field
	[INTENSE=	{**NORMAL**| DARK| BRIGHT}]	Specifies the light intensity of the constant field
	[MDT=	{Y|**N**}]	Controls the default setting of the modified data tag indicator
	[OUTLINE=	{**NOUTLINE**|BOX| [ORIGHT][OLEFT] [OOVER][OUNDER]}]	Specifies if lines are to be drawn at the edges of fields on DBCS devices
	[PROTECT=	{UNPROTECT| PROTECT| **ASKIP**}]	Specifies whether data can be entered into the field
	[.]		

COLOR=

MONO
Specifies monochrome (default value)

BLUE Specifies blue

RED Specifies red

PINK Specifies pink

GREEN
Specifies green

TURQUOISE
Specifies turquoise

YELLOW
Specifies yellow

WHITE
Specifies white

COLOR specifies the color of the text in the field when displayed on a color device.

CURSOR=

Y Specifies that the cursor is positioned on this field when the map is displayed

N Specifies that the cursor is not positioned in this field when the map is displayed

CURSOR specifies if the cursor is positioned on this field when the map is displayed. The cursor attribute can be overridden by the program using the SET map item CURSOR processing statement. The default is the first unprotected named variable field on the display.

DATA=

ALPHA
Specifies the field will accept character data (default value)

NUMERIC
Specifies the field will accept only numeric data

DATA specifies the data type for the field. Devices that support numeric shift will automatically shift to numeric mode when data is entered in this field if NUMERIC is specified.

DETECT=

Y Specifies that a light pen will cause a display device interrupt

N Specifies that a light pen will not cause a display device interrupt (default value)

DETECT specifies whether a light pen can be used to cause a display device interrupt in this field. If DETECT=Y and a light pen is pointed at the field (or it can be cursor selected on some devices), the program is notified.

ENTER=

Y Specifies data is required in this field before the Enter or action key is pressed

N Specifies data is not required in this field (default value)

ENTER specifies whether the device enforces required input entry.

FILL=

Y Specifies an execution error if a user does not enter enough characters to fill the field

N Specifies no error if less data is entered into a field than is required to fill the field (default value)

FILL specifies whether the device enforces filling an entire input field. This specification does not imply that input is required for a field. It means that if a user enters any characters in a field that has this attribute, the entire field must be filled.

HILITE=

NOHILITE
Specifies the field has no special highlighting (default value)

BLINK
Specifies the field flashes on and off when a map displays

RVIDEO
Specifies the field displays in reverse video—a dark or light background with contrasting light or dark text

USCORE
Specifies the field is underlined

INTENSE=

NORMAL
Specifies that text is displayed at normal light intensity (default value)

DARK
Specifies that text is not visible on a terminal display (passwords, for example)

BRIGHT
Specifies that text is displayed at higher or brighter light intensity than normal

INTENSE specifies the brightness of the value in the field when displayed on a screen.

MDT=

Y Specifies the MDT is set on for a field

N Specifies the MDT is set off for a field (default value)

MDT (modified data tag) controls the default setting of the modified data tag indicator for the field when the field is displayed.

If a field has MDT set on, default data can be presented to the user during execution. When the map is displayed during execution, the default data for the field is displayed; the user can accept that data and press the Enter key or enter new data over the default data. On import, if MDT=Y for a constant field, it is changed to N.

OUTLINE=

NOUTLINE
Draws no outline (default value)

BOX Draws all four outline lines

ORIGHT
Draws a vertical outline on the right side of the field

OLEFT
Draws a vertical outline on the left side of the field

OOVER
Draws a line between the previous row and the current row of the field

OUNDER
Draws a line between the current row and the next row of the field

OUTLINE allows lines to be drawn at the edges of fields on the DBCS devices. When using this attribute, you must specify either NOUTLINE, BOX, or any combination of the remaining values. This is the only attribute that applies to both displays and printers.

PROTECT=

UNPROTECT
Specifies that data can be entered in a field

PROTECT
Specifies that data cannot be entered in a field

ASKIP
(Autoskip) Specifies that the field is automatically skipped by the cursor when the tab key is used (default value)

PROTECT specifies whether data can be entered in a field. A field with ASKIP specified is also protected.

Variable field definition

The VFIELD tag specifies information about a variable field on the map. The optional tags MAPEDITS, MESSAGES, and VATTR provide additional information about a variable field. The EVFIELD tag closes the definition and is required for each VFIELD.

:VFIELD tag attributes

Syntax	**Attributes**		
:map ⋮	***:VFIELD***		
:vfield	**Attributes**	**Values**	**Uses**
⋮	BYTES=	*field length in bytes*	Specifies the number of positions the variable field occupies
:mapedits	COLUMN=	*column number*	Specifies the column of the byte immediately preceding the variable field
⋮	[DECIMALS=	{**0** | *decimal places*}]	Specifies the number of positions to the right of a decimal point in numeric items
:messages	[DESC=	*'field description'*]	Describes what the field represents
⋮	[EDITORDR=	*number*]	Specifies the position in the map edit sequence
:vattr	[INDEX=	*index value*]	Specifies the index value when the field is an element in an array
⋮	[NAME=	*field name*]	Identifies a map variable field
:evfield	ROW=	*row number*	Specifies the row of the byte immediately preceding the variable field
⋮	[TYPE=	{**CHA** | DBCS | MIX | NUM}]	Specifies the data type
:emap	[**.**[*initial value*]]		Specifies the initial value of the field
	:EVFIELD [.]		

BYTES=

field length in bytes
Specifies the number of positions the variable field occupies

COLUMN=

column number
Specifies the column of the byte immediately preceding the variable field

DECIMALS=

0 Specifies 0 decimal places (default value)

decimal places
Specifies a number of decimal places

DECIMALS is the number of positions to the right of an implied decimal point in numeric items. Decimal places can be specified only for numeric data. The maximum number of decimal positions is 18 or the number of digits defined for the field, whichever is smaller.

DESC=

'field description'
Describes what the variable field represents

DESC is a text description of what the variable field represents. The text can be up to 30 characters.

EDITORDR=

number
Specifies the position in the map edit sequence

EDITORDR allows you to modify the sequence in which map variable fields are checked on input to a program. If you specify EDITORDR for any variable named field on the map, you must specify it for all named fields, unless the

field is in an array. EDITORDR can only be specified for the first element of an array. If not specified for any variable field, the default edit order is the order in which the variable fields appear on the map.

INDEX=

index value
Specifies the index value when the field on the map is an element of an array

INDEX must be a number between one and 9999. When the index value is 1, any of the attributes of the VFIELD tag can be specified. If the field is part of an array, then INDEX must be specified.

If INDEX is greater than 1, the following attributes **can** be specified:
- ROW
- COLUMN
- BYTES
- TYPE
- NAME
- INITIAL VALUE.

If INDEX is greater than 1, the following attributes **cannot** be specified:
- DECIMALS
- DESC
- EDITORDR.

If INDEX is greater than 1, the following tags **cannot** be specified:
- MAPEDITS
- MESSAGES.

NAME=

field name
Identifies a map variable field

NAME consists of 1 to 32 characters and must meet the following conventions:
- The first character must be alphabetic or national (A-Z, $, #, @)
- The remaining characters must be alphanumeric or national characters, hyphens, or underscores (A-Z, 0-9, $, #, @, -, _).
- The name cannot contain blanks or have an EZE prefix, except EZEMSG.

The name can be a DBCS name up to 15 DBCS characters long with no embedded blanks.

ROW=

row number
Specifies the row of the byte immediately preceding the variable field

TYPE=

CHA Specifies any character data (default value)

DBCS Specifies double-byte character set, ideographic character data that requires two positions for each character

MIX Specifies DBCS and single-byte data in the same field

NUM Specifies numeric data

TYPE specifies the type of data that is acceptable for the variable field. Data input is checked to ensure the type of data entered is valid for the field.

initial value

initial value
Specifies the initial value for the variable field

Field text begins in the next available byte immediately following the period. The period is required if field text is supplied. If a line of text in the external source format file is longer than 71 characters, place a continuation character in column 72. Any character in column 72 is a continuation character. The continuation character causes concatenation of the two lines on import.

Map edit characteristic definition

The MAPEDITS tag defines a list of map editing characteristics that can be specified for variable fields described with the VFIELD tag. This optional tag and its attributes further describes the VFIELD tag.

:MAPEDITS tag attributes

Syntax	Attributes		
:map ⋮ ⋮	*:MAPEDITS*		
:vfield ⋮ ⋮	**Attributes** [CURRSYMB=	**Values** {Y | **N**}]	**Uses** Specifies whether the currency symbol is supported
:mapedits	[DATEFORM=	{*'date edit mask'* | SYSGREGRN | SYSJULIAN | *number*}]	Specifies the date format editing
⋮ ⋮	[EDITRTN=	*edit routine*]	Specifies a routine or edit table for special data editing
:messages	[FILLCHAR=	{'N' | *'fill character'*}]	Specifies the character used to fill unused field positions
⋮ ⋮	[FLDFOLD=	{MAP | Y | N}]	Specifies whether data entered into the field is folded
:vattr	[HEXEDIT=	{Y | **N**}]	Specifies whether only hexadecimal digits can be entered in the input field
⋮ ⋮	[INPUTREQ=	{Y | **N**}]	Specifies whether valid data must be entered
:evfield	[JUSTIFY=	{LEF | RIG | N}]	Specifies the position of data when it is shorter than the length of the field
⋮ ⋮	[MININPUT=	{**N** | *positions*}]	Specifies the minimum number of required characters
:emap	[NUMSEP=	{Y | **N**}]	Specifies whether data can contain numeric separators
	[RANGE=	*lowvalue highvalue*]	Specifies the range of valid numeric values
	[SIGN=	{LEA | TRA | N}]	Specifies a sign in the field as leading, trailing, or none
	[SOSI=	{Y | N}]	Specifies whether to check mixed data to see if it will fit into the field after conversion
	[ZEROEDIT=	{Y | **N**}]	Specifies the format for numeric fields that have zero values
	[.]		

CURRSYMB=

Y Specifies the currency symbol is supported

N Specifies the currency symbol is not supported (default value)

CURRSYMB indicates whether the currency symbol is supported in the field. Y is valid only for numeric fields. When defining field length, remember the currency symbol takes up one position.

DATEFORM=

'date edit mask'
Specifies a valid date edit mask

SYSGREGRN
Specifies a system default Gregorian format

SYSJULIAN
Specifies a system default Julian format

number
Specifies a predefined date format is used when data is entered or displayed

DATEFORM indicates the date format of a map field.

Date edit mask has a maximum length of 10 and consists of DD, MM, YY, DDD, or YYYY in any order separated by a non-numeric single-byte character, except D, M, and Y. If a single quote is used as a separator character, it must be specified as two single quotes in succession.

For all dates, the map field length must be the same length as the valid date edit mask. For example, for date edit mask 'MM/DD/YYYY' the map field length must be 10, while for 'MM/DD/YY' the map field length must be 8.

Number consists of an integer from 1 to 17 that represents a predefined date edit mask, as indicated in the following table.

Number	Short Format	Long Format
1	MM/DD/YY	MM/DD/YYYY
2	MM-DD-YY	MM-DD-YYYY
3	MM:YY	MM:YYYY

Number	Short Format	Long Format
4	YY/MM/DD	YYYY/MM/DD
5	YY-MM-DD	YYYY-MM-DD
6	YY:MM	YYYY:MM
7	DD/MM/YY	DD/MM/YYYY
8	DD-MM-YY	DD-MM-YYYY
9	DD:MM:YY	DD:MM:YYYY
10	YY-DDD	YYYY-DDD
11	YY:DDD	YYYY:DDD
12	MM.DD.YY	MM.DD.YYYY
13	YY.MM.DD	YYYY.MM.DD
14	DD.MM.YY	DD.MM.YYYY
15	YY.DDD	YYYY.DDD
16	SYSGREGRN	SYSGREGRN
17	SYSJULIAN	SYSJULIAN

Field edits should be defined as follows when specifying date edit option:
- CURRSYMB=N
- DECIMALS=0
- HEXEDIT=N
- NUMSEP=N
- SIGN=N.

The TYPE value must be CHA or NUM.

EDITRTN=

edit routine
Specifies a routine or edit table for special editing of data in a variable field

EDITRTN indicates the name of a routine or edit table used for special editing of data that a user enters in a variable field. You can specify one of the following:

- The name of one of the following types of editing tables:
  - Match valid table
  - Match invalid table
  - Range match valid table.

- The name of one of the following special function word subroutines:
  - Modulus 10 check digit routine (EZEC10)
  - Modulus 11 check digit routine (EZEC11).
- The name of a function used as an edit routine.

FILLCHAR=

'N' Specifies a null fill character

'fill character'
Specifies the character used to fill unused field positions

FILLCHAR indicates the character used to fill unused field positions on output to the terminal or printer. The default fill character is a null.

FLDFOLD

MAP Specifies folding on all character variable fields

Y Specifies that data is folded

N Specifies that data is not folded

FLDFOLD specifies whether lowercase alphabetic characters entered by the user are to be folded (converted) to uppercase. MAP means that all character variable fields and all single-byte data entered into mixed fields are folded when a user enters data into the fields. This field must be MAP if you specify VARFOLD=Y. If you specify VARFOLD=N, this value can be either Y or N. For more information, see VARFOLD= on page 159.

HEXEDIT=

Y Specifies that only hexadecimal digits can be entered

N Specifies the map field is not checked for hexadecimal characters (default value)

HEXEDIT specifies whether the input field is checked for hexadecimal digits. The data type of the variable field must be CHA for Y to be specified.

INPUTREQ=

Y	Specifies that data must be entered in the map field
N	Specifies that input is not required in the field (default value)

INPUTREQ indicates whether data must be entered in a map field. When a field contains data other than blanks for character type or zeros for numeric types, the field is considered to have input. Blanks in a character field and zero in a numeric field do not satisfy the input required edit check. Even if blanks and zeros are valid values, INPUTREQ returns an error message unless the user types data into the field.

JUSTIFY=

LEF	Specifies left justification (default for character data)
RIG	Specifies right justification (default for numeric data)
N	Specifies no justification

JUSTIFY specifies the position of data in a variable field when the data is shorter than the length of the field. If JUSTIFY is not specified, character data is left justified and numeric data is right justified. Right justification is required for numeric data with decimal positions or sign specified. JUSTIFY=N is not valid for numeric fields.

MININPUT=

N	Specifies no minimum number of characters required (default value)
positions	Specifies the minimum number of characters that must be entered in a valid variable field

MININPUT specifies the minimum number of characters that must be entered in a variable field if any data is entered.

NUMSEP=

Y Specifies that data can contain numeric separators

N Specifies that data cannot contain numeric separators (default value)

NUMSEP specifies whether data in a field can contain the previously defined numeric separator. The default numeric separator is a comma. Your system administrator can change the default character using the customization procedures for language-dependent options.

If YES is specified, numeric separators are allowed in the field on input. After input editing, the numeric separators are removed before the field is placed in internal storage. On output, numeric separators are inserted between every three significant digits. Every fourth position to the left of the decimal point will be a separator. You can only specify NUMSEP=Y for numeric fields. When defining field length, remember that each numeric separator takes up one position. If NUMSEP=N is specified, validation is done to ensure that the user did not enter a numeric separator in the field.

RANGE=

lowvalue
Specifies the smallest numeric value for a field

highvalue
Specifies the largest numeric value for a field

RANGE specifies the range of valid numeric values for a field. Lowvalue is the smallest numeric value that can be entered in a specified field. Highvalue is the largest numeric value that can be entered in a specified field. The high and low values must have the same length, number of decimal positions, and sign as defined for the field. Both low and high values must be specified if this attribute is used. The values are separated by a space and cannot be longer than the length of the field being defined. RANGE can be specified only for numeric fields.

SIGN=

LEA	Specifies a leading sign
TRA	Specifies a trailing sign
N	Specifies no sign (default value)

SIGN specifies whether a sign is displayed in a field and whether it is a leading or a trailing sign. You can specify signs for numeric fields only. When defining field length, remember the sign takes up one position.

SOSI=

Y	Specifies that mixed data entered in the map field on an ASCII device is to be checked to see if it will fit into the field after conversion into the EBCDIC SO/SI format for mixed data (default for mixed type fields)
N	Specifies that mixed data entered in the map field on an ASCII device is not to be checked to see if it will fit into the field after conversion into the EBCDIC SO/SI format for mixed data (default for all fields except mixed type fields)

ZEROEDIT=

Y	Specifies the display format for numeric fields containing 0 values
N	Specifies that no editing is done on 0 numeric fields (default value)

ZEROEDIT specifies the display format for numeric fields that have 0 values. The following table gives a list of what a numeric field will contain when ZEROEDIT is specified with either Y or N. The sample field has been defined as right justified with a length of 11. A "b" represents a blank fill character.

DEC (2)	CURRSYMB / NUMSEP	FILL	ZEROEDIT= N	ZEROEDIT= Y
N	N	N	nulls	0
N	N	b	blanks	bbbbbbbbbb0

DEC (2)	CURRSYMB / NUMSEP	FILL	ZEROEDIT= N	ZEROEDIT= Y
N	N	0	00000000000	00000000000
N	N	*	**********	**********0
Y	N	N	nulls	0.00
Y	N	b	blanks	bbbbbbb0.00
Y	N	0	00000000000	00000000.00
Y	N	*	**********	*******0.00
Y	Y	N	nulls	$0.00
Y	Y	b	blanks	bbbbbb$0.00
Y	Y	0	00000000000	$000,000.00
Y	Y	*	**********	******$0.00

Field Edit Message Definition

The MESSAGES tag specifies the user-defined messages for the field defined on the previous VFIELD tag that are generated when the corresponding field edits are called as specified on the MAPEDITS tag. This optional tag and its attributes further describes the VFIELD tag.

The message number specified must correspond with a number in the user-defined message table.

:MESSAGES Tag Attributes

Syntax	Attributes		
:map . . .	*:MESSAGES*		
:vfield . . .	**Attributes**	**Values**	**Uses**
	[EDITMSG=	*message number*]	Specifies the number of the edit routine error message
:mapedits	[INVALMSG=	*message number*]	Specifies the number of the data type error message
. . .	[MININMSG=	*message number*]	Specifies the number of the minimum input error message
:messages	[RANGEMSG=	*message number*]	Specifies the number of the value error message
. . .	[REQMSG=	*message number*]	Specifies the number of the input required error message

Syntax	Attributes		
:vattr : : :evfield : : :emap	[.]		

EDITMSG=

message number
Specifies the number of the message displayed when the data fails a modulus check or table edit check

If you do not specify this attribute, the default error message for a modulus check is “Modulus check error—reenter”, and the default error message for a table edit check is “Table edit validity error—reenter”.

INVALMSG=

message number
Specifies the number of the message displayed when the data entered is incompatible with the variable field data type

If you do not specify this attribute, the default error message for invalid data type is “Data type error in input—reenter”.

MININMSG=

message number
Specifies the number of the message displayed when the minimum input edit check fails

If you do not specify this attribute, the default error message for this error is “Input minimum length error—reenter”.

RANGEMSG=

message number
Specifies the number of the message displayed when the minimum or maximum value check fails

If you do not specify this attribute, the default error message is "Input not within defined range—reenter".

REQMSG=

message number
Specifies the number of the message displayed when the input required edit check fails

If you do not specify this attribute, the default error message for this error is "No input received for required field—reenter".

Variable Field Attribute Definition

The VATTR tag defines attributes for individual variable fields described in the VFIELD tag. This tag is optional.

:VATTR Tag Attributes

Syntax	**Attributes**		
:map . .	***:VATTR***		
:vfield	**Attributes**	**Values**	**Uses**
. .	[COLOR=	{**MONO** | BLUE | RED | PINK | GREEN | TURQUOISE | WHITE | YELLOW}]	Specifies the color of the text in the field
:mapedits	[CURSOR=	{Y | N}]	Identifies if the cursor is positioned on the field
. .	[DATA=	{**ALPHA** | NUMERIC}]	Specifies the field data type attribute
:messages	[DETECT=	{Y | **N**}]	Specifies whether the field is light pen detectable
. .	[ENTER=	{Y | **N**}]	Specifies if data is required to be entered into the field

Syntax	Attributes		
:vattr	[FILL=	{Y | **N**}]	Specifies whether an error occurs when a user does not enter enough characters to fill the field
⋮	[HILITE=	{**NOHILITE** | BLINK | RVIDEO | USCORE}]	Specifies highlighting characteristics of the variable field
:evfield	[INTENSE=	{**NORMAL** | DARK | BRIGHT}]	Specifies the light intensity of the variable field
⋮	[MDT=	{Y | **N**}]	Controls the default setting of the modified data tag indicator
:emap	[OUTLINE=	{**NOUTLINE** | BOX | [ORIGHT][OLEFT] [OOVER][OUNDER]}]	Specifies if lines are to be drawn at the edges of fields on DBCS devices
	[PROTECT=	{**UNPROTECT** | PROTECT | ASKIP}]	Specifies whether data can be entered into the field
	[.]		

COLOR=

MONO
Specifies monochrome (default value)

BLUE Specifies blue

RED Specifies red

PINK Specifies pink

GREEN
Specifies green

TURQUOISE
Specifies turquoise

YELLOW
Specifies yellow

WHITE
Specifies white

COLOR specifies the color of the text in the field when displayed on a color device.

CURSOR=

Y Specifies that the cursor is positioned on this field when the map is displayed

N Specifies that the cursor is not positioned in this field when the map is displayed (default value)

CURSOR specifies if the cursor is positioned on this field when the map is displayed. The cursor attribute can be overridden by the program using the SET map item CURSOR processing statement. If CURSOR=Y, the default cursor position is the first unprotected named variable field on the display.

DATA=

ALPHA
Specifies that the field will accept character data (default value)

NUMERIC
Specifies that the field will accept only numeric data

DATA specifies the data type for the field. Devices that support numeric shift will automatically shift to numeric mode when data is entered in this field if NUMERIC is specified.

DETECT=

Y Specifies that a light pen will cause a display device interrupt

N Specifies that a light pen will not cause a display device interrupt (default value)

DETECT specifies whether a light pen can be used to cause a display device interrupt in this field. If DETECT=Y and a light pen is pointed at the field (or it can be cursor selected on some devices), the program is notified.

ENTER=

Y Specifies data is required in this field before the Enter or an action key is pressed

N Specifies data is not required in this field (default value)

ENTER specifies whether the device enforces required input entry.

FILL=

Y Specifies an execution error if a user does not enter enough characters to fill the field

N Specifies no error if less data is entered into a field than is required to fill the field (default value)

FILL specifies whether the device enforces filling an entire input field. This specification does not imply that input is required for a field. It means that if a user enters any characters in a field that has this attribute, the entire field must be filled.

HILITE=

NOHILITE
Specifies the field has no special highlighting (default value)

BLINK
Specifies the field flashes on and off when a map displays

RVIDEO
Specifies the field displays in reverse video—a dark or light background with contrasting light or dark text

USCORE
Specifies the field is underlined

INTENSE=

NORMAL
Specifies that text is displayed at normal light intensity (default value)

DARK
Specifies that text is not visible on a terminal display (passwords, for example)

BRIGHT
Specifies that text is displayed at higher or brighter light intensity than normal

INTENSE specifies the brightness of the value in the field when displayed on a screen.

MDT=

Y Specifies the MDT is set on for a field

N Specifies the MDT is set off for a field (default value)

MDT (modified data tag) controls the default setting of the modified data indicator for the field when the field is displayed.

If a field has MDT set on, default data can be presented to the user during execution. When the map is displayed during execution, the definition data for the field is displayed; the user can accept that data and press Enter or enter new data over the default data. On import, if MDT=Y for a constant field, it is changed to N.

OUTLINE=

NOUTLINE
Draws no outline (default value)

BOX Draws all four outline lines

ORIGHT
Draws a vertical outline on the right side of the field

OLEFT
Draws a vertical outline on the left side of the field

OOVER
Draws a line between the previous row and the current row of the field

OUNDER
Draws a line between the current row and the next row of the field

OUTLINE allows lines to be drawn at the edges of fields on the DBCS devices. When using this attribute, you must specify either NOUTLINE, BOX, or any combination of the remaining values. This is the only attribute that applies to both displays and printers.

PROTECT=

UNPROTECT
Specifies that data can be entered in a field (default value)

PROTECT
Specifies that data cannot be entered in a field

ASKIP
(Autoskip) Specifies the field is automatically skipped

PROTECT specifies whether data can be entered in a field. A field with ASKIP specified is also protected.

Templates tracability information

Templates tracability information is defined for a map part as described in "Chapter 16. Templates traceability information structures" on page 213.

Map structure syntax example

```
:map       grpname  = xxxxxx   mapname  = xxxxxxxx
           date     = 'mm/dd/yyyy' time = 'hh:mm:ss'
           mapsize  = xxx xxx  startpos = xxx xxx  sosipos  = x
           helpmap  = xxxxxxxx helpkey  = xx
           bypkey   = xx xx xx xx xx
           devices  = xxxxxxx xxxxxxx xxxxxxx xxxxxxx xxxxxxx xxxxxxx
                      xxxxxxx xxxxxxx xxxxxxx xxxxxxx xxxxxxx xxxxxxx
                      xxxxxxx xxxxxxx xxxxxxx xxxxxxx xxxxxxx xxxxxxx
                      xxxxxxx xxxxxxx xxxxxxx xxxxxxx xxxxxxx xxxxxxx.
:present   varfold  = x    deffold  = x
           tabpos   = xxx xxx xxx xxx xxx xxx xxx xxx xxx xxx.
:cfield    row  = xxx   column = xxx
           type = xxxx  bytes  = xxxxx

.field text-------------------------------------------------------------X
field text continued---------------

:cattr     hilite   = xxxxxxxx  intense  = xxxxxx   protect = xxxxxxxxx
           color    = xxxxxxxxx data     = xxxxxxx  enter   = x
           mdt      = x         fill     = x        cursor  = x
           detect   = x         outline  = xxxxxxxx
:ecfield.
:vfield    row  = xxx   column = xxx
           type = xxxx  bytes  = xxxxx  decimals = xx
           name = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  index    = xxxx
           desc = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
           editordr = xxx
.initial value----------------------------------------------------------X
initial value continued------------
:mapedits  fldfold  = x
           range    = xxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxx
           mininput = xx
           sosi     = x
           fillchar = 'x'
           editrtn  = xxxxxxxxxxxxxxxxxx
           dateform = 'xxxxxxxxxx' currsymb = x   sign     = xxx
           numsep   = x
           inputreq = x            justify  = xxx hexedit  = x
           zeroedit = x.
:messages  invalmsg = xxxxx
           mininmsg = xxxxx
           reqmsg   = xxxxx
           editmsg  = xxxxx
           rangemsg = xxxxx.
:vattr     hilite   = xxxxxxxx  intense  = xxxxxx  protect  = xxxxxxxxx
           color    = xxxxxxxxx     data = xxxxxxx
           enter    = x  mdt = x  fill = x  cursor = x  detect = x
           outline  = xxxxxxxx.
:evfield.
:emap.
```

Note: This example illustrates the syntax of all map definition tags and attributes. Actual definitions do not require or use all tags and attributes shown here.

Chapter 10. Map group structures

This chapter describes the external source format tags and attributes that define a VisualAge Generator map group.

Map group definition

The MAPG tag defines attributes for individual map group parts. The AREA tag further defines aspects of a map group and can have attributes. A map group part defines the floating areas used by the maps in the map group.

The EMAPG tag is required to close the definition.

Note: The MAPG tag is required only for maps with floating areas.

:MAPG tag attributes

Syntax	**Attributes**		
:mapg ⋮	***:MAPG***		
:area ⋮ :vagt	**Attributes**	**Values**	**Uses**
	[DATE=	*'modification date'*]	Specifies the date the map group part was last modified
⋮	GRPNAME=	*name*	Identifies the map group
:tracbag	[TIME=	*'modification time'*]	Specifies the time the map group part was last modified
⋮	[.]		
:etracbag **:emapg**	***:EMAPG*** [.]		

DATE=

'modification date'
Specifies the date the map group part was last modified

DATE format is mm/dd/yyyy. Single quotes are optional.

GRPNAME=

name Identifies a group of maps used with a program

GRPNAME identifies a group of maps used with one program. GRPNAME consists of 1 to 6 characters and must meet the following conventions:

- The first character must be alphabetic (A-Z)
- The remaining characters must be alphanumeric (A-Z, 0-9)
- The name cannot contain blanks or have an EZE prefix.

Map group names in generated COBOL programs cannot contain the characters $, #, or @.

TIME=

'modification time'
Specifies the time the map group part was last modified

TIME format is hh:mm:ss. Single quotes are optional.

Floating area definition

The AREA tag defines a floating area. This required tag further describes the MAPG and EMAPG tag set. If multiple areas are specified, they must have unique devices specified.

:AREA tag attributes

Syntax	**Attributes**		
:mapg . .	***:AREA***		
:area . .	**Attributes**	**Values**	**Uses**
	DEVICE=	*device name*	Specifies the device associated with the floating area
:emapg	[SIZE=	*depth width*]	Specifies the size of the floating area
	[STARTPOS=	{**1 1** | *line column*}]	Specifies the starting position of the floating area
	[.]		

DEVICE=

device name
Specifies the device associated with the floating area

You can specify each device only once in the map group.

The following table gives the valid devices and their sizes.

Supported IBM Devices	Rows	Columns
3643-2	6	40
3277-1	12	40
3643-4	16	64
3278-1, 3278-1B, ANY-1D	12	80
3278-2, 3278-2B, ANY-2D	24	80
5550D (DBCS display device)	24	80
3278-3, 3278-3B, ANY-3D	32	80
3278-4, 3278-4B, ANY-4D	43	80
3278-5, 3278-5B, ANY-5D	27	132
ANY-D (3290 Configured as 62x160)	255	160
3767 PRINT-B PRINTER (physical size 66x132)	255	132
5550P (DBCS printer with size 66x158)	255	158

SIZE=

depth Specifies the depth of the floating area in number of lines

width Specifies the width of the floating area in number of columns

SIZE is the size of the floating area. Specify size in terms of depth (number of lines) and width (number of columns). Separate the depth and width values by a space.

STARTPOS=

1 Specifies line one (default value)

1 Specifies column one (default value)

line Specifies the line coordinate where the floating area starts

column
Specifies the column coordinate where the floating area starts

STARTPOS is the starting position for the floating area. Specify the starting position in terms of the beginning line and column. Separate the line and column values by a space.

Templates tracability information

Templates tracability information is defined for a map group part as described in "Chapter 16. Templates traceability information structures" on page 213.

Map group structure example

```
:mapg grpname  = xxxxxx
          date     = 'mm/dd/yyyy' time   = 'hh:mm:ss'.
:area     size     = xxx xxx  startpos = xxx xxx
          device   = xxxxxxx.
:emapg.
```

Note: This example illustrates the syntax of all map definition tags and attributes. Actual definitions do not require or use all tags and attributes shown here.

Chapter 11. Options file structures

This chapter describes the tags and attributes that define a VisualAge Generator options file part.

Options file definition

The OPTIONS tag defines attributes for individual options file parts. The TEXT tag further defines aspects of the options file part. The EOPTIONS tag closes the definition and is required.

:OPTIONS tag attributes

Syntax	**Attributes**		
:options	***:OPTIONS***		
	Attributes	**Values**	**Uses**
⋮	[DATE=	*'modification date'*]	Specifies the date the options file part was last modified
:text	NAME=	*'options file name'*	Specifies the name of the options file part
⋮	[TIME=	*'modification time'*]	Specifies the time the options file part was last modified
:etext	[.]		
⋮	***:EOPTIONS*** [.]		
:vagt			
⋮			
:tracbag			
⋮			
:etracbag			
:eoptions			

DATE=

modification date
Specifies the date the options file was last modified

DATE format is mm/dd/yyyy. Single quotation marks are optional.

NAME=

'options file part name'
Identifies the name of the options file part

Note: This attribute is not folded. In order for the part to be found during generation, the case of the referenced part name must exactly match the actual part name.

TIME=

modification time
Specifies the time the options file part was last modified

TIME format is hh:mm:ss. Single quotation marks are optional.

Text definition

The TEXT tag begins the character text that represents the options file part definition. The ETEXT tag closes the text area and is required.

:TEXT tag values

Syntax	**Usage**	
:options . .	***:TEXT***	
	Values	**Uses**
:text . .	[.[*Options file text*]] ***:ETEXT***[.]	Specifies the options file definition
:etext . .		
:eoptions		

Templates tracability information

Templates tracability information is defined for an options file part as described in "Chapter 16. Templates traceability information structures" on page 213.

Options file structure example

```
:options name  = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
          date     = 'mm/dd/yyyy' time   = 'hh:mm:ss'.
:text.options file text
:etext.
:eoptions.
```

Note: This example illustrates the syntax of all options file tags and attributes. Actual definitions do not require or use all tags and attributes shown here.

Chapter 12. Resource association file structures

This chapter describes the tags and attributes that define a VisualAge Generator resource association file part.

Resource association file definition

The RSRCS tag defines attributes for individual resource association file parts. The TEXT tag further defines aspects of the resource association file part. The ERSRCS tag closes the definition and is required.

:RSRCS tag attributes

Syntax	**Attributes**		
:rsrcs	***:RSRCS***		
	Attributes	**Values**	**Uses**
⋮	[DATE=	*'modification date'*]	Specifies the date the resource association file part was last modified
:text	NAME=	*'resource association file name'*	Specifies the name of the resource association file part
⋮	[TIME=	*'modification time'*]	Specifies the time the resource association file part was last modified
:etext	[.]		
⋮	***:ERSRCS*** [.]		
:vagt			
⋮			
:tracbag			
⋮			
:etracbag			
:ersrcs			

DATE=

modification date
Specifies the date the resource association file part was last modified

DATE format is mm/dd/yyyy. Single quotation marks are optional.

NAME=

resource association file part name
Identifies the name of the resource association file part

Note: This attribute is not folded. In order for the part to be found during generation, the case of the referenced part name must exactly match the actual part name.

TIME=

modification time
Specifies the time the resource association file part was last modified

TIME format is hh:mm:ss. Single quotation marks are optional.

Text definition

The TEXT tag begins the character text that represents the resource association file part definition. The ETEXT tag closes the text area and is required.

:TEXT tag values

Syntax	**Usage**	
:rsrcs . .	***:TEXT***	
	Values	**Uses**
:text	[.[*Resource association file text*]]	Specifies the resource association file definition
. .	***:ETEXT***[.]	

Syntax	Usage	
:etext . . . :ersrcs		

Note: The Resource association file text is not folded.

Templates tracability information

Templates tracability information is defined for a resource association file part as described in "Chapter 16. Templates traceability information structures" on page 213.

Resource association file structure example

```
:rsrcs   name  = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
           date     = 'mm/dd/yyyy' time   = 'hh:mm:ss'.
:text.resource association file text
:etext.
:ersrcs.
```

Note: This example illustrates the syntax of all resource association file tags and attributes. Actual definitions do not require or use all tags and attributes shown here.

Chapter 13. Linkage table file structures

This chapter describes the tags and attributes that define a VisualAge Generator linkage table file part.

Linkage table file definition

The LINKAGE tag defines attributes for individual linkage table file parts. The TEXT tag further defines aspects of the linkage table file part. The ELINKAGE tag closes the definition and is required.

:LINKAGE tag attributes

Syntax	Attributes		
:linkage	***:LINKAGE***		
	Attributes	**Values**	**Uses**
⋮	[DATE=	*'modification date'*]	Specifies the date the linkage table file part was last modified
:text	NAME=	*'linkage table file name'*	Specifies the name of the linkage table file part
⋮	[TIME=	*'modification time'*]	Specifies the time the linkage table file part was last modified
:etext	[.]		
⋮	***:ELINKAGE*** [.]		
:elinkage			

DATE=

modification date
Specifies the date the linkage table file part was last modified

DATE format is mm/dd/yyyy. Single quotation marks are optional.

NAME=

'linkage table file part name'
Identifies the name of the linkage table file part

Note: This attribute is not folded. In order for the part to be found during generation, the case of the referenced part name must exactly match the actual part name.

TIME=

modification time
Specifies the time the linkage table file part was last modified

TIME format is hh:mm:ss. Single quotation marks are optional.

Text definition

The TEXT tag begins the character text that represents the linkage table file part definition. The ETEXT tag closes the text area and is required.

:TEXT tag values

Syntax	Usage	
:linkage . . .	***:TEXT***	
	Values	**Uses**
:text	[.[*Linkage table file text*]]	Specifies the linkage table file definition
. . .	***:ETEXT***[.]	
:etext . . .		
:elinkage		

Templates tracability information

Templates tracability information is defined for a linkage table file part as described in "Chapter 16. Templates traceability information structures" on page 213.

Linkage table file structure example

```
:linkage name  = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
          date     = 'mm/dd/yyyy' time   = 'hh:mm:ss'.
:text.linkage table file text
:etext.
:elinkage.
```

Note: This example illustrates the syntax of all linkage table file tags and attributes. Actual definitions do not require or use all tags and attributes shown here.

Chapter 14. Bind control file structures

This chapter describes the tags and attributes that define a VisualAge Generator bind control file part.

Bind control file definition

The BNDCTRL tag defines attributes for individual bind control file parts. The TEXT tag further defines aspects of the bind control file part. The EBNDCTRL tag closes the definition and is required.

:BNDCTRL tag attributes

Syntax	**Attributes**		
:bndctrl	***:BNDCTRL***		
	Attributes	**Values**	**Uses**
⋮	[DATE=	*'modification date'*]	Specifies the date the bind control file part was last modified
:text	NAME=	*'bind control file name'*	Specifies the name of the bind control file part
⋮	[TIME=	*'modification time'*]	Specifies the time the bind control file part was last modified
:etext	[.]		
⋮	***:EBNDCTRL*** [.]		
:vagt			
⋮			
:tracbag			
⋮			
:etracbag			
:ebndctrl			

DATE=

modification date
Specifies the date the bind control file part was last modified

DATE format is mm/dd/yyyy. Single quotation marks are optional.

NAME=

'bind control file part name'
Identifies the name of the bind control file part

Note: This attribute is not folded. In order for the part to be found during generation, the case of the referenced part name must exactly match the actual part name.

TIME=

modification time
Specifies the time the bind control file part was last modified

TIME format is hh:mm:ss. Single quotation marks are optional.

Text definition

The TEXT tag begins the character text that represents the bind control file part definition. The ETEXT tag closes the text area and is required.

:TEXT tag values

Syntax	**Usage**	
:bndctrl : :	***:TEXT***	
	Values	**Uses**
:text	[.[*Bind control file text*]]	Specifies the bind control file definition
: :	***:ETEXT***[.]	
:etext : :		
:ebndctrl		

Templates tracability information

Templates tracability information is defined for a bind control file part as described in "Chapter 16. Templates traceability information structures" on page 213.

Bind control file structure example

```
:bndctrl name  = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
           date     = 'mm/dd/yyyy' time   = 'hh:mm:ss'.
:text.bind control file text
:etext.
:ebndctrl.
```

Note: This example illustrates the syntax of all bind control file tags and attributes. Actual definitions do not require or use all tags and attributes shown here.

Chapter 15. Link edit file structures

This chapter describes the tags and attributes that define a VisualAge Generator link edit file part.

Link edit file definition

The LNKEDIT tag defines attributes for individual link edit file parts. The TEXT tag further defines aspects of the link edit file part. The ELNKEDIT tag closes the definition and is required.

:LNKEDIT tag attributes

Syntax	**Attributes**		
:lnkedit	***:LNKEDIT***		
	Attributes	**Values**	**Uses**
⋮	[DATE=	*'modification date'*]	Specifies the date the link edit file part was last modified
:text	NAME=	*'link edit file name'*	Specifies the name of the link edit file part
⋮	[TIME=	*'modification time'*]	Specifies the time the link edit file part was last modified
:etext	[.]		
⋮	***:ELNKEDIT*** [.]		
:vagt			
⋮			
:tracbag			
⋮			
:etracbag			
:elnkedit			

DATE=

modification date
Specifies the date the link edit file part was last modified

DATE format is mm/dd/yyyy. Single quotation marks are optional.

NAME=

'link edit file part name'
Identifies the name of the link edit file part

Note: This attribute is not folded. In order for the part to be found during generation, the case of the referenced part name must exactly match the actual part name.

TIME=

modification time
Specifies the time the link edit file part was last modified

TIME format is hh:mm:ss. Single quotation marks are optional.

Text definition

The TEXT tag begins the character text that represents the link edit file part definition. The ETEXT tag closes the text area and is required.

:TEXT tag values

Syntax	**Usage**	
:lnkedit : :	***:TEXT***	
	Values	**Uses**
:text : :	[.[*Link edit file text*]] ***:ETEXT***[.]	Specifies the link edit file definition
:etext : :		
:elnkedit		

Templates tracability information

Templates tracability information is defined for a link edit file part as described in "Chapter 16. Templates traceability information structures" on page 213.

Link edit file structure example

```
:lnkedit name  = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
          date     = 'mm/dd/yyyy' time   = 'hh:mm:ss'.
:text.link edit file text
:etext.
:elnkedit.
```

Note: This example illustrates the syntax of all link edit file tags and attributes. Actual definitions do not require or use all tags and attributes shown here.

Chapter 16. Templates traceability information structures

This chapter describes the external source format tags and attributes that define the traceability information that can be associated with any VisualAge Generator part. These tags exist only in Templates generated 4GL parts. They contain information about the VAGT source definition (the instance from which the 4GL part has been generated by Templates), and information about the contents of the 4GL parts when it was generated. This information enables VAGTemplates 'smart regeneration' capabilities.

For each part, the :VAGT :TRACBAG, and :ETRACBAG tags are the last tags that can appear before the tag to end the part.

Templates traceability information definition

:VAGT tag attributes

Syntax	**Attributes**		
:vagt ⋮	***:VAGT***		
	Attributes	**Values**	**Uses**
:tracbag	[PARMVERS=	'*time stamp*']	Specifies when the VAGT generation parameter was last modified
⋮	[TRACECAT=	{ **NONE** | API | HOOK | INTERNAL | RAD}]	Specifies the trace category
:etracbag	[VISITNAM=	'*visitable name*']	Specifies the name of the VAGT source definition
	[VISITTYP=	{BUSINESSOBJECT | DATAELEMENT | INTERFACEUNIT | RELATIONALTABLE | VALUESTYPE | **WORKSPACE**}]	Specifies the entity type of the VAGT source definition
	[VISITVER=	'*time stamp*']	Specifies when the VAGT source definition was last modified
	[WSPCNAME=	'*workspace name*']	Specifies the name of the VAGT workspace definition

Syntax	Attributes		
	[WSPCVERS=	*'time stamp'*]	Specifies when the VAGT workspace definition was last modified
	[.]		

PARMVERS=

'time stamp'
Specifies the timestamp of the generation parameter associated with the VAGT definition. This generation parameter object has been used to generate the current 4GL part.

PARMVERS format is yyyy-mm-dd-hh.mm.ss.

TRACECAT=

NONE
Indicates that no trace category is specified. (default value)

API Specifies that the 4GL part can be used but will be regenerated by Templates

HOOK
Specifies that the 4GL part is a place of customization and will not be overriden if not asked

INTERNAL
Specifies that the 4GL part should not be used and not customized. It is generated for internal purposes only

RAD Specifies that the 4GL part is a place of customization and will not be overriden if not asked

Note: For VisualAge Generator parts, RAD and HOOK components are equivalent.

VISITNAM=

'visitable name'
Specifies the name of the VAGT definition. It is from this definition and the generation parameter object that the current 4GL part has been generated.

VISITTYP=

VISITTYP is the entity type of the VAGT definition. The current 4GL part has been generated from an instance of this type.

BUSINESSOBJECT
The entity type is a Business Object.

DATAELEMENT
The entity type is a Data Element.

INTERFACEUNIT
The entity type is an Interface Unit.

RELATIONALTABLE
The entity type is an Relational Table.

VALUESTYPE
The entity type is a Value Style.

WORKSPACE
The entity type is a Workspace. (default value)

VISITVER=

'time stamp'
Specifies the timestamp of the VAGT source instance.

VISITVER format is yyyy-mm-dd-hh.mm.ss.

WSPCNAME=

'workspace name'
Specifies the name of the workspace, if relevant, that has been used during the generation of the source instance.

WSPCVERS=

'time stamp'
Specifies the timestamp of the VAGT workspace definition.

WSPCVERS format is yyyy-mm-dd-hh.mm.ss.

Trace bag information

The TRACBAG tag begins the code that represents the traceability information for the contents of the 4GL part, used by VisualAge Generator Templates. This optional tag further defines the tag set for each VisualAge Generator part. The ETRACBAG tag closes the definition, and if you use the TRACBAG tag, the ETRACBAG tag is required.

The TRACBAG tag can appear only once within a part definition.

:TRACBAG tag values

Syntax	Usage	
:vagt . . .	*:TRACBAG*	
:tracbag . . .	**Values** [.[*trace bag information*]]	**Uses** Specifies the traceability information needed for VisualAge Generator Templates. It describes the composition of the 4GL part when it was generated.
:etracbag	*:ETRACBAG* [.]	

Only the library size and machine size limit the number of lines that you can specify. Each line can contain up to 2000 bytes. You can use both uppercase and lowercase characters. The trace information must be preceeded by a period.

Templates traceability information syntax example

```
:vagt   visittyp = Business Object
           tracecat = API
           vistinam = 'MyBusinessObject'
           visitver = 'yyyy-mm-dd-hh.mm.ss'
           wspcname = 'MyWorkspace'
           wspcname = 'yyyy-mm-dd-hh.mm.ss'
           parmvers = 'yyyy-mm-dd-hh.mm.ss'
:tracbag.tracebag text
:etracbag
```

Note: This example illustrates the syntax of all Templates traceability information tags and attributes. Actual definitions do not require or use all tags and attributes shown here.

Appendix A. External source format functions with VisualAge Generator commands

This appendix introduces the VisualAge Generator commands that control external source format functions.

Import and export of VisualAge Generator parts

External source format files are serial files. On import, the external source format provides the necessary defaults for tags and attributes that are optional and not specified. Export produces all tags and attributes, except in the following cases:

- If the data item and the map field map editing attributes are default values
- If the SQL clauses are unmodified default values
- If the DL/I SSAs are unmodified default values

After using external source format to define your programs and parts, use the IMPORT subcommand to import from a serial file to a library and use the EXPORT subcommand to export from a library to a serial file.

External source format file

In VisualAge Generator, the first 72 columns within each record contain library definitions in the external source format. Columns 73-80 may contain a sequence number that is ignored on import. The format of definitions within the external source format file varies depending on the library part type.

Note: The following parts do not observe the 72 column limit and do not contain sequence numbers:

- GUI parts in interchange format
- Logic definition statements and SQL statements
- Options file structures
- Resource Association file structures
- Linkage Table file structures
- Bind Control file structures
- Link Edit file structures
- Record help for UI records
- Title for UI records
- Field help for UI records
- Label for UI records
- Field help for UI items

- Label for UI items

In VisualAge Generator Developer, the external source format file is a serial file. Variable length records are accepted on import.

Header record

VisualAge Generator produces a new header each time an export is requested. A single file might contain multiple headers. On the batch command, the default is to append to the end of the file unless you specify the /REPLACE option.

The following table shows the format of the first 50 bytes of the first record of an external source format file. For any external source format files that do not have the header, VisualAge Generator Developer (VGD) returns message HPT.PE.282.e, "The external source format file cannot be imported because it does not contain a valid external source format header line."

Field description	Data type	Length (bytes)
Identifier and meaning **:EZEE** External source format files	Character	6
Version number and product release **322** CSP 3.2.2 **330** CSP 3.3.0 **410** CSP 4.1; CSP2AD 1.0 and 1.1; VGD 1.0, 1.1, 2.0, and 2.2 **430** VGD 3.0 and 3.1 **440** VGD 4.0	Character	3
Reserved	Character	1
Reserved for enabled products	Character	12
Reserved	Character	1
File creation date - mm/dd/yy	Character	8
Reserved	Character	1
File creation time - hh:mm:ss	Character	8
Reserved	Character	10

Error processing

In VisualAge Generator Developer, the VisualAge Generator IMPORT subcommand validates the external source format parts individually to prevent invalid parts from getting in the library. However, parts in the same import file are not validated against each other. When multiple parts in the

external source format file have the same name, only one of them remains in the library. Which part remains depends on the options you specify. If you specify /dupapp=none, the first valid part in the external source format file is imported provided no parts by that name already exist in the library before import begins. If you specify any other value for the /dupapp option, the last valid part in the file imported by that name is the one that remains in the library.

When invalid parts are in the import file, you receive an error message. Error messages are written to STDOUT.

Import does not compare definitions to be imported with parts that exist in the library. This can cause errors. For example, if the part you import is already in the library as an EXECUTE or file input/output function, and you replace the imported part with a CONVERSE function, then any batch program in the library that uses the function is now in error.

Conditions detected by the test facility or by the preprocessor are not checked for on import and do not prevent the IMPORT. For example, IMPORT does not detect the following:

- Statements that refer to parts of the wrong part types
- Function objects that do not match the I/O option
- Map edit routines that cause screen input/output

Conditions detected by program, data, or map definition are verified on import and prevent the import. For a complete list of the detected error conditions for VisualAge Generator Developer, refer to the *VisualAge Generator Messages and Problem Determination Guide*, and especially see the messages with prefixes of HPT.PE and HPT.PL. Some examples are as follows:

- Invalid syntax in processing statements
- Invalid level numbers in record structures
- Hex edit only valid for character fields on a map
- Maximum number of occurs exceeded.

Prologue lines that are too long are an example of a condition that is corrected on IMPORT and does not prevent IMPORT. They are split into multiple lines if they are too long.

Appendix B. DBCS support

The HPTRULES.NLS file contains the set of national characters for all supported language versions of VisualAge Generator. For information on EZERDEV.NLS, refer to the *VisualAge Generator Installation Guide* document.

Tags and attributes are always exported in lowercase.

Note: The information below for DBCS data applies only when exporting and importing on a DBCS device. When exporting or importing on a single-byte device, DBCS data is treated the same as single-byte data.

Whenever DBCS data is produced in the external source format file, VisualAge Generator inserts the necessary shift-out (SO: X'0E') and shift-in (SI: X'0F') characters into the boundaries between SBCS and DBCS data, so that:

- The file can be displayed on an EBCDIC DBCS-supported device
- The file has a common definition form across all IBM platforms.

On import, VisualAge Generator verifies the external source format file and ensures the following:

- DBCS and mixed data are legal DBCS or mixed strings
- SO and SI characters are present and placed in correct positions on either EBCDIC systems or ASCII systems
- Pairs of SO and SI characters are completed within a single line.

Note: When planning for NLS and translation, remember that the $, #, @, ¬, and ^ characters are not in the National Language syntactic character set and might not be represented by equivalent code points across different code pages. Avoid using these characters if your program will be transferred between machines with differing code pages. This is particularly true in transfers between System/370 host machines and workstations. An alternative for handling codepage-dependent characters is to use the hptcnv30 utility. See *Migration Guide* for more information.

For DBCS constant and variable fields and for table contents (but not for literals in statements), the SO and SI characters are stripped from the data before it is placed in the library. For mixed constant and variable fields, for table contents, and for comments (but not for literals in statements), contiguous SO and SI characters that appear at the end of one line and at the beginning of the next are stripped from the data before it is placed in the

library. This allows extra SO and SI characters to be added to the data in the external source format file when data must be split across multiple lines.

On the workstation, when importing on a DBCS device, SO and SI characters are stripped from DBCS and mixed data before storing.

Figure 1 shows several ways to split data across multiple lines.

When splitting data to fill column 71 of the external source format file causes a DBCS character to be split in half, place an SI character in column 71, a continuation character in column 72, and the rest of the data on the next line (with an SO character in column 1). See **1** in Figure 1.

When column 71 is filled with the second half of a DBCS character, put an SI character in column 72 (it has the double duty of being an SI character and a continuation character) and put the rest of the data on the next line (with an SO character in column 1). See **2** in Figure 1.

When you want a contiguous SO and SI character set in the data and the data must be split between the SO and the SI characters, put an extra pair of SI and SO characters in the external source format file. See **3** in Figure 1.

```
         External Source Format file     Data as stored in the library
         1     6         7         8
1234567890 ... 012345678901234567890
   1           za<DiDjDkDl>x             za<DiDjDkDlDmDnDo>ab
<DmDnDo>ab
   2           a<DiDjDkDlDm>             a<DiDjDkDlDmDnDo>abc
<DnDo>abc
   3           za<DiDjDkDl>x             za<DiDjDkDl><DmDnDo>
<><DmDnDo>

where <, >, and Dx mean SO, SI, and a DBCS character, respectively.
```

Figure 1. External Source Format Data Storage

Appendix C. Tags not supported by VisualAge Generator

This appendix lists, by structure type, obsolete external source format tags, attributes, and attribute values from the Cross System Product. Substituted attributes and values are listed where applicable other tags are accepted by VisualAge Generator for compatibility with Cross System Product.

Program structures

- The APPL tag is replaced by the PROGRAM tag.
- The MAINPRC tag is replaced by the MAINFUN tag.
- MSGTABLE is equivalent to the MSGFILE attribute. Either MSGFILE or MSGTABLE may be specified but not both.
- The EXECMODE attribute is moved from the GENOPTS tag tothe APPL tag. The following rules apply to the EXECMODE attribute value:
  - If the EXECMODE attribute is specified on both the APPL and the GENOPTS tag, the value specified on the APPL tag is used. The value on the GENOPTS tag is ignored.
  - If the EXECMODE attribute is specified on the GENOPTS tag only, the value on the GENOPTS tag is used. On export, the EXECMODE attribute appears on the APPL tag.
  - An EXECMODE value of EITHER is accepted only on the GENOPTS tag but is changed to one of the valid values. If EITHER is specified for the EXECMODE attribute on the GENOPTS tag, the value is changed to SEGMENTED for a MAIN transaction. Otherwise, it is changed to NONSEGMENTED.
- The following GENOPTS attributes are supported to provide compatibility with Cross System Product and to help you migrate your programs to VisualAge Generator. Use the external source format file conversion utility to extract the generation options from an external source format file into a generation options file. Refer to the *Migrating Cross System Product Applications to VisualAge Generator* document for more information on this utility.
  - ANSIGEN
  - ANSIMOD
  - CLIST
  - CREATREF
  - DB2GEN
  - DB2MOD
  - DDSGEN
  - FOLD

  - GENGRP1
  - GENGRP2
  - JOBNAME
  - LINKAGE
  - LINES
  - LOADLIB
  - MAPGRP1
  - MAPGRP2
  - OPTIONS
  - PRINT
  - SEGTRAN
  - SQLBLOCK
  - SQLGEN
  - SQLID
  - SQLMOD
  - VALIDLOC
  - VALIDSQL
- TARGSYS and its attribute, SYSTEM, are obsolete and ignored on import.
- GENFILE and its attributes, FILENAME, FILETYPE, PCBNO, SYSNAME, and SYSTEM, are obsolete and ignored on import.
- GENTABLE and its attributes, KEEP, NAME, and TABLEGEN are obsolete and ignored on import.
- HTFFILE and its attributes, HOSTTRAN, HTFERITM, HTFWKITM, and ASYNC are obsolete and ignored on import.
- The COMPONENT attribute is obsolete and ignored on import.

GUI client structures

- The COMPT attribute is obsolete and ignored on import.

Process structures

- The PROCESS tag is migrated to the FUNC tag on import.
- The COMPONENT attribute is obsolete and ignored on import.
- The UPDPROC attribute is replaced by the UPDFUNC attribute.

Statement group structures

- The GROUP tag is migrated to the FUNC tag on import.
- The COMPONENT attribute is obsolete and ignored on import.

Record structures

- The FILELOC attribute on the RECORD tag is obsolete and is ignored on import.
- The CREATOR and TBLENAME attributes are no longer required to uniquely identify a table in a relational database. The TABLEID attribute now serves this function. If CREATOR and TBLENAME are used, their values will be concatenated (separated by a period) and combined into a value for the TABLEID attribute.
- The COMPONENT attribute is obsolete and ignored on import.

Table structures

- The following GENOPTS attributes are supported to provide compatibility with Cross System Product. They are ignored on import:
  - FOLD
  - LOADLIB
  - JOBNAME
  - LINES
  - OPTIONS
  - PRINT
  - TYPEUSE
- The COMPONENT attribute is obsolete and ignored on import.
- The ASSOCTO attribute is obsolete and ignored on import.

Data item structures

- The COMPT attribute is obsolete and ignored on import.
- The ASSOCTO attribute is obsolete and ignored on import.

Program specification block structures

- The COMPT attribute is obsolete and ignored on import.
- The ASSOCTO attribute is obsolete and ignored on import.

Map structures

- The DEVICES values 8775-1C, 8775-2C, 8775-3C, and 8775-4C are obsolete and ignored on import. If only DEVICES values are specified, the following substitutions occur:
  - 8775-1C is changed to ANY-1D
  - 8775-2C is changed to ANY-2D
  - 8775-3C is changed to ANY-3D
  - 8775-4C is changed to ANY-4D.
- The following attributes are obsolete and ignored on import:

- COMPT
- CONSTANT
- DBCSCONS
- DBCSVAR
- MIXCONS
- MIXVAR
- SPACER
- VARIABLE

Map group structures

- The DEVICE values 8775-1C, 8775-2C, 8775-3C, and 8775-4C are obsolete and ignored on import. If only DEVICE values are specified, the following substitutions occur:
  - 8775-1C is changed to ANY-1D
  - 8775-2C is changed to ANY-2D
  - 8775-3C is changed to ANY-3D
  - 8775-4C is changed to ANY-4D.
- The COMPT attribute is obsolete and ignored on import.

Index

A

B

C

D

E

F

G

H

I

J

K

Q

R

S

T

U

V

W

Z

Readers' Comments — We'd Like to Hear from You

VisualAge Generator
External Source Format Reference
Version 4.5

Publication No. SH23-0265-01

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
SH23-0265-01

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

SH23-0265-01

Cut or Fold
Along Line

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SH23-0265-01