VisualAge Generator

Programmer's Reference

Version 4.5

SH23-0262-01

VisualAge Generator

Programmer's Reference

Version 4.5

SH23-0262-01

Note

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

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.

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:

ACF/VTAM
AD/Cycle
AIX
AS/400
C Set ++
CICS
CICS OS/2
CICS/ESA
CICS/MVS
CICS/VSE
COBOL/2
COBOL/370
COBOL/400
DataJoiner
DB2
DB2/2
DB2/400
DB2/6000
DRDA
FAA
GDDM
IBM
IBMLink
IMS
IMS/ESA
InfoExplorer
Language Environment
MVS
MVS/ESA
Operating System/2
OS/2
OS/400
RACF
RS/6000
SAA
SQL/DS
SQL/400
System/370
TeamConnection

Virtual Machine/Enterprise Systems Architecture
VisualAge
VisualGen
VM/ESA

The following terms are trademarks of other companies:

Adobe	Adobe Systems Incorporated
HP-UX	Hewlett-Packard Company
Micro Focus IMS Option	Micro Focus Limited

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

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

About this document

You can use this document as a reference for writing VisualAge Generator programs. If you are writing a program, you can use this document to look up language element syntax, usage, and examples. This document, along with *VisualAge Generator Design Guide, VisualAge for Smalltalk User's Guide, and VisualAge Generator Generation Guide* , serves as a source of general information needed to write a VisualAge Generator program.

Another use is as a design guide for portable programs. If you are designing a VisualAge Generator program to run in an environment other than the environment in which the program is written, you can use this document to become aware of the portability or compatibility considerations that could affect the way your program runs.

The language elements described in this document are grouped into non-procedural elements (Part Specifications) and procedural elements (Scripting Language, also known as "code").

Part specifications are grouped by the following part types:

Graphical user interface specification
An event-driven program that contains the graphical user interface of a program.

Program specification
Text or 3270 user interface program, batch program, or server program.

Function specification
An I/O operation or sequence of associated code used within a program.

Record specification
A data structure representing temporary working storage or a file or database record.

Table specification
A data array containing a set of predefined values.

Program specification block specification
Definition of the hierarchical record relationships between DL/I record segments.

Item specification
A data element definition. The element can be part of a record or table.

Map specification
Definition of a text or 3270 user interface format or a printer format. Elements related to the map as a whole.

Map field specification
Elements related to individual constant and variable fields on the map.

Scripting language elements are used when you enter procedural logic (code) associated with function parts, or program flow statements that control the order in which the main (top-level) functions of a program run.

The scripting language elements are grouped by the following topics:

Program statements
VisualAge Generator language statements and syntax.

Special function words
VisualAge Generator defined variables and services.

Services
System services specific to some run-time environments.

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.

Part 1. VAGen parts

Chapter 1. Graphical user interfaces

A graphical user interface (GUI) program is an event-driven program that contains one or more windows that represent the graphical user interface through which the program user enters data and requests the actions performed by the program. GUI programs consist of GUI windows and the logic (functions) and data (records, tables) parts associated with the window. The GUI program calls batch or server programs to access files and databases.

Defining a GUI program requires an approach that is different from defining character-based programs. A VisualAge Generator GUI program requires that you visually construct the user interface of the program and that you also visually construct the communication between the visual parts of the program and the nonvisual logic and data parts.

Graphical user interfaces in Smalltalk

The VisualAge Smalltalk product ships a parts palette that includes templates for creating many visual and nonvisual program parts. VisualAge Generator ships the following extensions to the VisualAge Smalltalk parts palette:

- Additional features (such as actions, attributes and events) for parts shipped with VisualAge Smalltalk
- Additional VisualAge Generator categories and their parts

All of the VisualAge Generator extensions to the parts palette have names that begin with the **VAGen** prefix. The VisualAge Generator extensions are described in this chapter, along with some techniques for visual programming. The basic parts palette shipped with VisualAge Smalltalk is described in the VisualAge Smalltalk online help and the *VisualAge Smalltalk User's Reference*.

For more information on visual programming and the visual parts of a GUI program, refer to the *VisualAge Smalltalk User's Reference*. For information on defining nonvisual parts, refer to the other chapters in this book.

VisualAge Generator parts category for Smalltalk

The VisualAge Generator product ships categories and parts that are added to the VisualAge Smalltalk parts palette during installation. The following are the VisualAge Generator parts.

- VAGen Record
- VAGen Table
- VAGen Program

- VAGen Function

VAGen Data parts

VAGen Data Parts include VisualAge Generator Developer data parts you can use to help build a GUI client. The parts in this category are nonvisual. The parts are described in the following sections.

Notes:

1. The Settings view of an embedded view does not show promoted attributes that belong to VAGen Data Parts.
2. The Public Interface Editor's Promote Feature page cannot be used to promote VAGen Data Parts attributes that are associated with data items. To promote these attributes, choose the **Promote Part Feature...** option from the parts in the Composition Editor. Once the attributes are promoted, the Promote Feature page can be used to view them.

VAGen Record part: Select the **VAGen Record** part to add a VisualAge Generator record to the free-form surface.

VAGen Record attributes:

self The *self* attribute represents the part itself.

Connecting to this property of a record part to pass parameter to a function part expecting a record or to pass a record part into another part that contains a place holder for it (a VAGen variable).

data The *data* attribute represents the contents of the record, table, or data item.

Note: In the case of an occurs item, it represents an Ordered Collection with the values of the valid elements of the occurs item.

data item attributes and data item data attributes

Two attributes are created for each of the individual and top-level substructured data items in the record. One attribute represents the data item and the other attribute represents the data item data.

VAGen Record actions:

destroyPart

The *destroyPart* action destroys the part and its children, and releases all associated resources.

VAGen Record events:

destroyedPart
The *destroyedPart* event signals that the part and its children have been destroyed, and any system resources associated with them have been released.

You can use any of the data item attributes in the record as a source of an event-to-action connection. The event is the modification of the value of the data item when the application runs. For example, you can use a flag in a working storage record and set the flag from the VisualAge Generator logic to trigger an action, such as opening or closing a window, in the application.

VAGen Record properties: There are no settable properties for this part.

VAGen Table part: Select the **VAGen Table** part to add VisualAge Generator tables to the free-form surface.

VAGen Table attributes:

self The *self* attribute represents the part itself.

Connecting to this property of a record part to pass parameter to a function part expecting a record or to pass a record part into another part that contains a place holder for it (a VAGen variable).

data The *data* attribute represents the contents of the record, table, or data item.

Note: In the case of an occurs item, it represents an ordered collection with the values of the valid elements of the occurs item.

table columns
The *table columns* attribute represents the data items that make up the columns of a table.

VAGen Table actions:

destroyPart
The *destroyPart* action destroys the part and its children, and releases all associated resources.

VAGen Table events:

destroyedPart
The *destroyedPart* event signals that the part and its children have been destroyed, and any system resources associated with them have been released.

VAGen Table properties: There are no settable properties for this part.

VAGen Logic parts

VAGen Logic Parts include VisualAge Generator logic parts you can use in building a GUI program. These parts are nonvisual. The parts are described in the following sections.

VAGen Program part: Select the **VAGen Program** part to add a VisualAge Generator program or a non-VisualAge Generator program to the free-form surface. You can use the VAGen Program part as a way to visually call server programs.

VAGen Program attributes:

self The *self* attribute represents the part itself.

Torn-off attributes get their values by using the *self* attribute. In general, it is a read-only attribute, passing a value or values to another part but not receiving any values.

You can use the *self* attribute of a torn-off attribute in an attribute-to-attribute connection.

You can use the *self* attribute as a parameter in an event-to-action connection.

lastResult

The *lastResult* attribute represents an object of the type HptProgramResult. The *lastResult* attribute is the result of the last call to this VAGen Callable Function.

The HptProgramResult object type is defined with the following features:

- **Attributes**

  returnCode

  The *returnCode* attribute represents an Integer value that is the return code value of the VAGen Program. In the case of a local DLL call, the return code is the value returned by the function. In the case of a remote call, the value is a VAGen middleware Reason Code.

- **Actions**

  displayError

  The *displayError* action displays the VAGen middleware error message that resulted from a remote call. The failure event of the VAGen Program can be connected to this action so that an error message will be displayed when the remote call fails.

getErrorText

The *getErrorText* action returns the VAGen middleware error message that resulted from a remote call.

linkageInfo

The *linkageInfo* attribute represents an object of the type HptCallLinkageInformation, and contains necessary linkage information for making a call to another function or program.

The HptCallLinkageInformation class is defined with the following attributes:

appType

The *appType* attribute specifies the remote application type. *appType* can have one of the following values:

VG The called program is a generated VisualAge Generator application. An additional parameter is automatically passed to the server to allow the server to return an error code to the middleware if the server application ends abnormally.

NON_VG

The called program was developed using a tool other than VisualAge Generator. Only the parameters passed on the call are passed to the called program.

conversionTable

The *conversionTable* attribute specifies the name of the conversion table used to perform automatic data conversion on the call to the remote application. The name is a 9-byte character array containing a null-terminated character string.

Some names have a special meaning:

***** Conversion is performed on the client using the default conversion table. You must enclose the asterisk in single quotes.

On OS/2, AIX, and Windows systems, the default is the conversion table specified in environment variable EZERCVT. If EZERCVT is not specified, the default is conversion table ELACN*xxx* (OS/2 or AIX) or ELACW*xxx* (Windows), where *xxx* is the national language code specified in environment variable EZERNLS. If EZERNLS is not specified, the default national language code is ENU.

BINARY

Only binary fields are converted. The byte order in the binary field is reversed.

This table is used with OS/2 and Windows clients communicating with AIX servers, and vice versa, when both the client and the server are running under the same code page.

NONE
No conversion is performed.

externalName
The *externalName* attribute specifies the name of the entry point in the DLL named in the library. The *externalName* value is ignored if *isRemote* is **true**.

isRemote
The *isRemote* attribute is a Boolean value that specifies if a call is to a remote or local function.

is32Bit
The *is32Bit* attribute is a Boolean value that tells whether a called DLL function is a 32 Bit or 16 Bit function. The *is32Bit* attribute is ignored if *isRemote* is **true**.

library
If *isRemote* is **true**, the *library* attribute specifies the name of the library that contains function to be called.

If *externalName* and *programName* are empty, *library* will also be the function name.

If *isRemote* is **true**, the *library* attribute specifies the OS/400 program library name. The name is a 20-byte character array containing a null-terminated character string. This value is used only with the Client Access/400 and Java400 protocols. It specifies the name of the OS/400 library that contains the called program. The default value is the application name if the array contains a null string.

linkageTableName
The *linkageTableName* attribute specifies the file name of the linkage table to be used if run-time bind is specified for the Protocol parameter.

If not specified (null string), the linkage table file name is obtained from environment variable CSOLINKTBL.

If the name is not fully qualified, the VisualAge Generator middleware uses the current DPATH (for OS/2) or PATH (for Windows) search path to find the table.

location
The *location* attribute specifies the protocol-dependent server

system name. The name is a 20–byte character array containing a null-terminated character string.

The following table shows the meaning of the identifier by protocol and the default value if a name is not specified (null string).

Protocol	Meaning of location	Default value
CICS DPL	CICS system identifier	System identifier defined for **applname** in the CICS tables.
CICSCLIENT	CICS system identifier	First system identifier specified in the CICS client initialization file.
DCE, DCESECURE, DCECICS, DCEIMS, DCEVM	Location where the server advertises in the DCE CDS database. The location is specified in the configuration file used when the VisualAge Generator DCE server program is started.	No default.
APPCIMS	CPIC side information identifier. The side information specifies: • Partner LU Alias • Transaction Program Name • Mode Name	No default
VG	See VisualAge Generator routing table description	Host defined for **applname in routing table**
TCPIP	TCP/IP hostname	No default
NPIPE	For remote NPIPE support (IBM's LAN Server), specify the COMPUTERNAME value from the LAN server's IBMLAN.INI file. For local NPIPE support, specify LOCAL.	No default
CA/400	AS/400 system identifier	The managing OS/400 system

luwType

The *luwType* attribute specifies the logical unit of work type. Values are as follows:

CLIENT

Unit of work is under client control.

Server updates are not committed or rolled back until the client requests commit or rollback using the EZECOMIT or EZEROLLB services of VisualAge

Generator or the *commit* or *rollback* actions of the VisualAge Generator *commSession* attribute for the class on whose free-form surface this part was dropped. Server applications cannot request commit or rollback.

Environments which do not support client-controlled unit of work will ignore this value.

SERVER
Server unit of work is independent of the client's unit of work. Commit (or rollback on abnormal termination) is automatically issued when the server returns. Server applications can request rollback.

parmform
The *parmform* attribute specifies the parameter format.

This option is supported only when calling through the CICS OS/2 ECI or CICS Client ECI. It is ignored for all other types of middleware.

Possible values for *parmform* are as follows:

COMMPTR
The server program expects to be called using the CSP/AE parameter-passing convention that uses pointers in the COMMAREA. Use only with MVS CICS or VSE CICS server programs that were generated or coded to use this parameter-passing convention.

COMMDATA
The server program expects to receive the parameter values in the CICS COMMAREA. The parameter values passed on the call are moved into a single buffer, each value adjoining the previous value without regard for boundary alignment. On return from the remote call, the values returned in the output buffer are moved back to the corresponding parameters that were passed on the call.

programName
The *programName* attribute specifies the name of the server program that is being called.

The name is a null-terminated character string with a maximum length of eight characters plus the null terminator.

protocol

The *protocol* attribute specifies the communications protocol used to communicate with the client application.

Valid values are as follows:

Runtime Bind
The communications protocol is read from the linkage table at run time. In addition, the following option values are read from the linkage table and any corresponding option specified in the *linkageInfo* settings is ignored:
- luwType
- appType
- parmform
- conversionTable
- location
- serverId
- library

CICS Client ECI
CICS Client External Call Interface

Client Access/400
Client Access/400

Java400
Java driver to connect to AS/400 system

APPC to IMS
LU 6.2 connection to IMS message processing region

DCE RPC Secure
Distributed Computing Environment Remote Procedure Call (DCE RPC), no authorization checking

DCE RPC Secure
Distributed Computing Environment Remote Procedure Call (DCE RPC) with authorization checking

DCE to CICS
Distributed Computing Environment to CICS

DCE to IMS
Distributed Computing Environment to IMS message processing region

DCE to VM
Distributed Computing Environment to VM

LU2 Logical Unit 2

Name Pipes
Name Pipes

PACBASE
PACBASE

TCP/IP
Transmission Control Protocol/Internetwork Protocol

serverId
The *serverId* attribute specifies the protocol-dependent server channel or transaction name. The name is a 20–byte character array containing a null-terminated character string.

The following table shows the meaning of the identifier by protocol and the default value if a name is not specified (null string).

Protocol	Meaning of Server Identifier
CICS, CICSCLIENT	Name of CICS transaction for the server. If client unit of work is specified, all applications called in the same unit of work must have the same server identifier. The default is the CICS server system mirror transaction.
DCE, DCESECURE	Server ID name advertised by the server in the DCE CDS database. The *serverId* is specified in the configuration file used when the VisualAge Generator DCE server program is started.

user-defined parameters
Parameters added to the public interface by selecting **Build parameters from definition** or **Add parameter** from the VAGen Program's pop-up menu.

VAGen Program actions:

destroyPart
The *destroyPart* action destroys the part and its children, and releases all associated resources.

execute
The *execute* action runs the function or program. This method accepts parameters for the program or function on the connection.

executeDeferred:
The *executeDeferred:* action runs the function or program after the specified delay. This method accepts parameters for the program or function on the connection. The argument to this action is the delay interval, which is specified in milliseconds. It is recommended that you do not use delays that are less than 100 milliseconds.

This action can be used to create a polling loop to wait on a certain resource to become available. The VAGen Logic part can check the resource and reschedule itself to executeDeferred again using a perform request structure. Once the resource is available, the loop can be terminated by simply not redispatching the VAGen Logic part again. Note that during the delay period, you are free to interact with the user interface and you can schedule other logic events to run prior to the timer expiration.

This action creates a background delay that will expire at the end of the delay interval. At that time, the logic part will be put on the execution queue to be processed at the next opportunity the queue is read. This means that there is no guarantee that the logic you have deferred will execute in the same order or at a set timer interval. The interval simply specifies when it would run at the earliest.

executeWithArguments:
The *executeWithArguments:* action calls the function or program, with the arguments given. This action requires an OrderedCollection of arguments to pass to the function or program being called.

executeDeferred:withArguments:
The *executeDeferred:withArguments:* action calls the function or program on the connection after the specified delay, with the arguments given. This action requires an OrderedCollection of arguments to pass to the function or program being called.

VAGen Program events:

destroyedPart
The *destroyedPart* event signals that the part and its children have been destroyed, and any system resources associated with them have been released.

failure
The *failure* event signals that the call has failed. The *lastResult* attribute is signaled with this event.

hasExecuted
The *hasExecuted* event signals that a function or program has been run.

success
The *success* event signals that the call was successful. The *lastResult* attribute is signaled with this event.

VAGen Program properties: The following VAGen Program attributes can be set from the **Settings** window for this part:

- lastResult

- linkageInfo

VAGen Function part: Select the **VAGen Function** part to add a VisualAge Generator function to the free-form surface.

VAGen Function attributes:

self The *self* attribute represents the part itself.

Torn-off attributes get their values by using the *self* attribute. In general, it is a read-only attribute, passing a value or values to another part but not receiving any values.

You can use the *self* attribute of a torn-off attribute in an attribute-to-attribute connection.

You can use the *self* attribute as a parameter in an event-to-action connection.

returnValue

The *returnValue* attribute is the result of the last call to a VAGen Function. It is the value returned on the EZERTN statement within the function.

The return values from the function mapped to the actual type of object returned by the returnValue attribute are:

any numeric value with no decimal precision
Integer

any numeric value with decimal precision
Fraction

Char HptChaString

DBCS HptDBCSString

Mixed HptMixString

Hex HptMixString

VAGen Function actions:

destroyPart

The *destroyPart* action destroys the part and its children, and releases all associated resources.

execute

The *execute* action runs the function or program. This method accepts parameters for the program or function on the connection. This feature might require parameter connections equivalent to the number of arguments the function part expects.

A value is returned if the returnValue attribute is defined for the function.

executeDeferred:

The *executeDeferred:* action runs the function or program after the specified delay. This method accepts parameters for the program or function on the connection. This feature might require parameter connections equivalent to the number of arguments the function part expects.The argument to this action is the delay interval, which is specified in milliseconds. It is recommended that you do not use delays that are less than 100 milliseconds.

This action can be used to create a polling loop to wait on a certain resource to become available. The VAGen Logic part can check the resource and reschedule itself to executeDeferred again using a perform request structure. Once the resource is available, the loop can be terminated by simply not redispatching the VAGen Logic part again. Note that during the delay period, you are free to interact with the user interface and you can schedule other logic events to run prior to the timer expiration.

This action creates a background delay that will expire at the end of the delay interval. At that time, the logic part will be put on the execution queue to be processed at the next opportunity the queue is read. This means that there is no guarantee that the logic you have deferred will execute in the same order or at a set timer interval. The interval simply specifies when it would run at the earliest.

executeWithArguments:

The *executeWithArguments:* action calls the function or program, with the arguments given. This action requires an OrderedCollection of arguments to pass to the function or program being called.

executeDeferred:withArguments:

The *executeDeferred:withArguments:* action calls the function or program on the connection after the specified delay, with the arguments given. This action requires an OrderedCollection of arguments to pass to the function or program being called.

VAGen Function events:

destroyedPart

The *destroyedPart* event signals that the part and its children have been destroyed, and any system resources associated with them have been released.

hasExecuted

The *hasExecuted* event signals that a function or program has been run.

VAGen Function properties: There are no settable properties for this part.

VAGen Container Details part

Select the **VAGen Container Details** part to add a part that displays information in rows and columns, with each item occupying a row. You can add columns by dragging a **Container Details Column** part from the parts palette. The number of rows is determined at run time by the object connected to the *items* attribute.

You can use a VAGen Container Details part to display the contents of a VisualAge Generator Table or an occurs item of a VisualAge Generator Record. You can also retrieve rows in packets that can be specified in the VAGen Container Details part, rather than retrieving all rows at once.

You can allow users of your application to change the contents of a cell in the VAGen Container Details part. They can do this by selecting the cell and entering data. It is recommended that you switch the *selectionPolicy* to single cell selection if you enable users to modify cells.

Part: VAGen Container Details

Class Name:
HptContainerDetailsView

Differences between Container Details and VAGen Container Details: The VAGen Container Details part provides the packeting support that is provided in the Packeting Container Details part. The packet support allows you to have individual rows retrieved in packets as requested by the VAGen Container Details part instead of having all rows retrieved at once. Besides the packet support, the VAGen Container Details part provides some other useful features, such as the events *cellValueChanged* and *userInputConvertError.*

VAGen Container Details inherits features from Container Details. For descriptions of the Container Details features, refer to the *VisualAge Smalltalk Reference*. VAGen Container Details has all the features provided by Container Details, as well as the following features:

attributes

- *packet*
- *packetEnabled*
- *packetSize*
- *totalRows*

actions

- *forcePacketRequest*
- *getSelectedCell*
- *getSelectedColumnIndices*
- *getTopIndex*

events

- *cellValueChanged*
- *packetRequested*
- *userInputConvertError*

VAGen Container Details attributes: The following VAGen Container Details attributes can be set as properties, which are available from the **Settings** view for this part:

packet The *packet* property represents the packet data structure that is used during packeting. It replaces the variable you previously had to connect from the packetRequested event. It supports the following features:

startRow
The row where the packet begins

endRow
The row where the packet ends

dataRows
The items or rows in the packet

packetEnabled
The *packetEnabled* attribute specifies whether the part will request data one packet at a time.

packetSize
The *packetSize* specifies the size of packets the VAGen Container Details part retrieves.

The *packetSize* attribute enables you to set the packet size before the packet request is run. This value is used as a suggested value when the packet is requested. However, if the suggested value is not valid, it is not updated automatically to reflect the actual size of the packet that was requested. To get the actual size of the packet that was requested, you should look at the *endRow* setting of the *packet* attribute.

totalRows
The *totalRows* attribute specifies the total number of rows within the container.

VAGen Container Details actions:

forcePacketRequest:
The *forcePacketRequest:* action triggers the packet request operation.

getSelectedCell
The *getSelectedCell* action returns a Point object representing the (column,row) coordinate of the selected cell in the container.

getSelectedColumnIndices
The *getSelectedColumnIndices* action returns a collection of the indices of the selected columns.

getTopIndex
The *getTopIndex* action returns the index of the top visible row in the container.

VAGen Container Details events:

cellValueChanged
The *cellValueChanged* event signals that one of the cells in the part was modified. This event contains a parameter that contains the following features:

- row
- column
- oldValue
- newValue

packetRequested
The *packetRequested* event signals that the part needs a new packet of information.

userInputConvertError
The *userInputConvertError* event signals that the user has typed an invalid value into the current cell of the part. This event is signalled with a cell error callback data object of type HptCellErrorCallbackData that contains the information about the cell that is in error. This callback data object has the following attributes:

newValue
This is the new string that was typed into the cell. The string is not valid for this cell. If this attribute is set to another valid string when this event is being handled, the valid value will be put in the cell. If this attribute is left unchanged when this event is handled, the last valid value of this cell (*oldValue*) will be put back in the cell.

The following steps show an example using the *userInputConvertError* event:

1. Connect the VAGen Container Details view's *userInputConvertError* event to the *self* attribute of a variable. When an error occurs, the variable will hold the cell error callback data object that is signalled with this event.
2. Connect the VAGen Container Details view's *userInputConvertError* event to the *prompt* action of a text prompter, to prompt the user for a valid value.
3. Connect the *answerString* attribute of the text prompter to the *newValue* attribute of the variable. You will need to connect to an unlisted attribute of the variable. When prompted for the unlisted attribute name, type **newValue**.
4. These connections will cause a text prompter to be displayed when an invalid value is typed into one of the cells and focus is then moved elsewhere. If a valid value is typed into the prompter, when **OK** is pressed, that valid value is entered into the error cell and committed. If an invalid value is typed, the prompter remains displayed until either a valid value is typed or **Cancel** is pressed. If **Cancel** is pressed, the last valid value in that cell is entered into the cell.

   Note: For a valid value to be put back into the cell, the *newValue* attribute must be set in the process of handling the *userInputConvertError* event. After the event is handled, setting *newValue* will not have any effect.

VAGen Variable part

Select the VAGen Variable part to enable your application to work with a part that is created at run time. A variable is a placeholder for the actual part, much like a parameter in an ordinary programming language.

When you add a variable to the free-form surface, you specify its class and connect the variable so that, at run time, it receives its identity from a part elsewhere in your application. At run time, a part of that class takes the place of the variable.

Part: VAGen Variable

Class Name:
AbtVariable

Refer to the Variable part description in the *VisualAge Smalltalk User's Reference* for more information on the Variable part.

Differences between VAGen Variable and Variable parts: The VAGen Variable is intended to be used with other VAGen parts. There are two main differences between VAGen Variable and Variable.

- When an attribute of a VAGen Variable is connected to another part's attribute or a connection's parameter, the VAGen Variable performs the appropriate conversion of its attribute into the expected type at the other end of the connection.

  For example, if you have a VAGen Variable that holds a VAGen Num data item, and you connect *data* attribute of the variable to *enabled* of a push button, the VAGen Variable converts the value of its *data* attribute from an integer or a number to a boolean as expected by *enabled* so that the alignment of the connection will be successful. If you use a VisualAge Smalltalk Variable instead of a VAGen Variable, this alignment will cause an error. Because of this automatic conversion, it is recommended that you use a VAGen Variable when the value that it holds is a VAGen Data part or is related to a VAGen Data part. When you tear off an attribute from a VAGen Data part, a VAGen Variable is used automatically for the tearoff.
- Because VAGen parts are not Smalltalk classes, you cannot change a VisualAge Smalltalk Variable's type to a VAGen part to have the appropriate features listed at edit time. Therefore, the VAGen Variable has an option to **Change VAGen Type** instead of **Change Type** for VisualAge Smalltalk Variable parts. **Change VAGen Type** allows you to specify at edit time the VAGen part that the VAGen Variable will hold at run time, and that allows the VAGen Variable part to show all features that are available for that VAGen part.

  Remember that VAGen Variables are placeholders for other parts, much like parameters in 3GL programming languages. Therefore, you must connect the *self* attribute of the VAGen Variable to identify the attributes of the part the VAGen Variable will receive at run time.

  Selecting **Change VAGen Type** is not sufficient to identify the part the VAGen Variable will receive. This function is available at edit time to make it easier to make connections to the features of a VAGen Variable.

VAGen Variable inherits from Variable. Therefore, it has the same features as Variable.

VAGen Variable attributes:

self The *self* attribute represents the part itself.

valueHolder
The *valueHolder* attribute holds the value of the variable or class.

After you select **Change VAGen Type** from the VAGen Variable's pop-up menu, the attributes of that part are available.

VAGen Variable actions: After you select **Change VAGen Type** from the VAGen Variable's pop-up menu, the actions of that part are available.

VAGen Variable events: After you select **Change VAGen Type** from the VAGen Variable's pop-up menu, the events of that part are available.

VAGen File Accessor part

Select the **File Accessor** part to add a part that will allow your application to manipulate text files. With this part, applications can read text files, display their contents in a string-capable control (for example, a Multi-line Edit), work with the actual file string, or save the string to any file and invoke the File dialog.

Use the **File Accessor** part to allow programs to manipulate text files. With this part, programs can read text files, display their contents in a string capable control, work with the actual file string, or save the string to any file and invoke the File dialog.

Part: VAGen File Accessor

Class Name:
HptFilePart

VAGen File Accessor attributes:

buffer The *buffer* attribute represents a String value that is a copy in memory of the contents of the file. The contents of this buffer are preserved when *fileSpec* is changed, which allows for save as operations. You can connect this attribute to a Multi-line Edit and be able to modify a file simply by using the *read* and *write* actions.

The initial value for this attribute is set on the **Settings** window for this part.

fileSpec
The *fileSpec* attribute represents a String value that contains the fully qualified file name. This is likely to be platform-specific. The File Accessor part always uses this attribute to indicate what file it is managing.

The initial value for this attribute is set on the **Settings** window for this part.

self The *self* attribute represents the part itself.

VAGen File Accessor actions:

read The *read* action clears the buffer, opens, reads the file into the buffer, and closes the file specified by the *fileSpec* attribute. If *fileSpec* is empty, the file selection dialog is invoked to prompt the user for the file name.

selectFile
The *selectFile* action invokes the platform-specific file selection dialog. The specified file is returned in *fileSpec*.

write The *write* action opens, writes the contents of the buffer to the file specified by *fileSpec*, and closes the file. If *fileSpec* is empty, the file selection dialog is invoked to prompt for the file name. To be able to invoke a save as type of operation, simply set *fileSpec* to the new file name prior to invoking the *write* action.

VAGen File Accessor events: There are no events for this part.

VAGen File Accessor properties: The following VAGen File Accessor properties can be set from the **Settings** view for this part:

- buffer
- fileSpec
- lastError

Additional VisualAge Generator features for VisualAge Smalltalk parts

The VisualAge Generator product ships additional attributes and actions for several VisualAge Smalltalk parts. The following sections describe these additional features and list all the VisualAge Smalltalk parts for which they are available.

Dynamically programming visual parts

VisualAge Generator provides several actions that enable you to dynamically build visual parts at run time. You can use these features to add new pages to a notebook part based on some logic, add new menu choices to a menu, or add new menu pull-downs.

You can use these features to establish a child-parent relationship between two parts. You can dynamically create a new instance of a visual part with the Object Factory part, and it can be displayed by the *openWidget* action without adding it to a parent part. However, if it does not have a parent part, it is treated as a top-level part and it is destroyed when it is closed. To prevent it from being destroyed, add it to a parent part using the *subpartNamed:* action.

A sample, partadr.dat, showing the use of these features is shipped with VisualAge Generator. For more information on the sample application, refer to the *VisualAge Generator Getting Started* document.

The following actions enable you to dynamically program visual parts. These actions require that an instance of a part that you want to add exists to be used as the "part" parameter. To create an instance of a part, use the Object Factory part.

- Adding a subpart to an existing composite visual part:

  VAGen subpartNamed:put:
  The *VAGen subpartNamed:put:* action adds a subpart to a composite part using the specified name. The part is added invisibly and must be opened using openWidget to be visible.

  VAGen subpartNamed:putOpened:
  The *VAGen subpartNamed:putOpened:* action adds a subpart to a composite part using the specified name. The part is added visibly so no openWidget is necessary after this action.

  VAGen subpartNamed:put:beforePartNamed:
  The *VAGen subpartNamed:put:beforePartNamed:* action adds a subpart invisibly at a specific order in the components list of the parent part. Note that this action will not show the part in the correct order in its parent unless the parent part has not been opened yet. If the parent is already opened, it must be closed and reopened to display its components in the correct order. If no before part name is specified, the part is added to the end of the list.

  VAGen subpartNamed:putOpened:beforePartNamed:
  The *VAGen subpartNamed:putOpened:beforePartNamed:* action adds a visible subpart at a specific order in the components list of the parent part. If no before part name is specified, the part is added to the end of the list. Though this action will add the part in the correct component sequence, it is likely the sequence will not appear correct until the parent is closed and reopened.

- Retrieving and destroying a subpart from a composite part:

  VAGen destroySubpartNamed:
  The *VAGen destroySubpartNamed:* action finds the subpart with the specified name and removes it from its parent.

  VAGen subpartNamed:
  The *VAGen subpartNamed:* action finds the subpart with the specified name. This search recursively scans all subparts of the composite part until a match is found. The result of this action can be assigned to a variable that can then act as the subpart.

- Showing and positioning an existing visual part:

  openWidget
  The *openWidget* action opens a modeless Window with respect to its parent Window.

closeWidget
The *closeWidget* action closes the part.

VAGen setX
The *VAGen setX* action positions the left edge of the widget at the specified outset.

VAGen setY
The *VAGen setY* action positions the top edge of the widget at the specified outset.

VAGen setHeight
The *VAGen setHeight* action sets the height of the widget.

VAGen setWidth
The *VAGen setWidth* action sets the width of the widget.

The positioning actions affect the framingSpec of the widget. However, they only allow you to set absolute/relative dimensions, not proportional or attachment settings. To access the more powerful attachment functions, you should use the *framingSpec* attribute.

The features are available with specific visual parts described in the sections below.

Nonvisual parts of class: AbtAppBldrPart and visual parts of class: AbtAppBldrView

Attributes: The attributes and actions described in this section apply to all nonvisual parts in the AbtAppBldrPart class and all visual parts in the AbtAppBldrView class.

VAGen commSession
The *VAGen commSession* attribute is a read-only attribute that is initialized to an object of type CmSession when the first call to a VAGen server is issued from the VAGen Callable Function or from a VAGen CALL statement in a function.

The CmSession class is defined with the following actions:

commit
When executed, the *commit* action propagates a commit to the server platforms, as appropriate. The commit will only have an effect when calls to servers using this session were made through the CICS Client or Client Access/400 middleware products and client unit of work was specified. In all other cases, *commit* results in an immediate return with no action taken.

rollback
When executed, the *rollback* action propagates a rollback to the server platforms, as appropriate. The rollback will only have an effect when calls to servers using this session were made through the CICS Client or Client Access/400 middleware products and client unit of work was specified. In all other cases, *rollback* results in an immediate return with no action taken.

If *VAGen inheritsCommSession* is set to true, then *VAGen commSession* is only set if the parent of the part is nil and the *VAGen commSessionOwner* attribute is nil. *VAGen commSessionOwner* is first checked for the session and if it is nil, then the parent is checked. If both are nil, then the *VAGen commSession* attribute is set to a new instance of the VAGen Communications Session part.

The *VAGen commSession* attribute can be torn off and used visually or it can be accessed through Smalltalk code.

VAGen commSessionOwner
The *VAGen commSessionOwner* attribute can be set to any instance of a subclass of *AbtAppBldrNonVisual*. *VAGen commSessionOwner* is used in conjunction with *VAGen inheritsCommSession* to control the ownership of a given communications session within a hierarchy of parts. If *VAGen commSessionOwner* is set, and *VAGen inheritsCommSession* is set to true, the session object will be looked for in that part.

VAGen inheritsCommSession
VAGen inheritsCommSession is a boolean attribute that controls how to look for the instance of a VAGen Communications Session part when a call to a VAGen server is issued. If set to true, the *VAGen commSessionOwner* is checked first. If the *VAGen commSessionOwner* is set to nil, then the parent is checked. If the *VAGen inheritsCommSession* attribute is false, then the current part returns the session that is stored in *VAGen commSession*.

Actions:

VAGen destroyTopLevelSubpartNamed:
The *VAGen destroyTopLevelSubpartNamed:* action finds the subpart with the specified name, removes it from its parent and destroys it. This action differs from the *VAGen destroySubpartNamed:* action provided by the primary part in that it starts the search from this part rather than the primary part.

VAGen topLevelSubpartNamed:
The *VAGen topLevelSubpartNamed:* action finds the subpart with the specified name. This search recursively scans all the subparts of the

composite part until a match is found. The result of this action can be assigned to a variable that can then act as the subpart. This action differs from the *VAGen subpartNamed:* action provided by the primary part in that it starts the search from this part rather than the primary part.

VAGen topLevelSubpartNamed:put:
The *VAGen topLevelSubpartNamed:put:* action adds a subpart to a composite part using the specified name. The part is added invisibly and must be opened using openWidget to be visible. This action differs from the *VAGen subpartNamed:put:* action provided by the primary part in that it adds the subpart to this part instead of adding the subpart to the primary part.

VAGen performRequest:
The *VAGen performRequest:* action executes actions stored in its parameter object, such as a record's data item.

For example, use the *VAGen performRequest:* action when you would like to conditionally trigger an action in the GUI program based on a computation that is performed in a VisualAge Generator logic part.

Visual parts of class: AbtBasicView

The actions described in this section apply to all visual parts in the class AbtBasicView in the following categories:

- Buttons
- Data Entry
- Lists
- Menus
- Canvas
- OS/2

Actions:

VAGen setHeight
The *VAGen setHeight* action sets the height of the widget.

VAGen setWidth
The *VAGen setWidth* action sets the width of the widget.

VAGen setX
The *VAGen setX* action positions the left edge of the widget at the specified offset.

VAGen setY
The *VAGen setY* action positions the top edge of the widget at the specified offset.

Visual parts that can contain other visual parts

The actions described in this section apply to all visual parts that can contain other visual parts, including:

- Container Details part (Class: AbtContainerDetailsView)
- Composite Views part (Class: AbtCompositeView)
- Parts in the Canvas category

Actions:

VAGen destroySubpartNamed:
The *VAGen destroySubpartNamed:* action finds the subpart with the specified name, removes it from its parent and destroys it.

VAGen subpartNamed:
The *VAGen subpartNamed:* action finds the subpart with the specified name. This search recursively scans all the subparts of the composite part until a match is found. The result of this action can be assigned to a variable that can then act as the subpart.

VAGen subpartNamed:put:
The *VAGen subpartNamed:put:* action adds a subpart to a composite part using the specified name. The part is added invisibly and must be opened using openWidget to be visible.

VAGen subpartNamed:put:beforePartNamed:
The *VAGen subpartNamed:put:beforePartNamed:* action adds a subpart invisibly at a specific order in the components list of the parent part. Note that this action will not show the part in the correct order in its parent unless the parent part has not been opened yet. If the parent is already opened, it must be closed and reopened to display its components in the correct order. If no before part name is specified, the part is added to the end of the list.

VAGen subpartNamed:putOpened:
The *VAGen subpartNamed:putOpened:* action adds a subpart to a composite part using the specified name. The part is added visibly, so no openWidget is necessary after this action.

VAGen subpartNamed:putOpened:beforePartNamed:
The *VAGen subpartNamed:putOpened:beforePartNamed:* action adds a visible subpart at a specific order in the components list of the parent part. If no before part name is specified, the part is added to the end of the list. Though this action will add the part in the correct component sequence, it is likely the sequence will not appear correct until the parent is closed and reopened.

Form, Group Box and Window parts

The attributes described in this section apply to the following parts:

- Form part (Class: AbtFormView)
- Group Box part (Class: AbtGroupBoxView)
- Window part (Class: AbtShellView)

Attributes:

VAGen topLevelEnabled

The *VAGen topLevelEnabled* attribute is a Boolean value that represents whether a part is available for user interaction. *VAGen topLevelEnabled* is different from *enabled* in that when *VAGen topLevelEnabled* is false for a part, it does not show the part's children as disabled. The children of the part will appear enabled.

Window part

The actions described in this section apply to the Window part of class AbtShellView.

Actions:

VAGen cancelCloseRequest:

The *VAGen cancelCloseRequest:* action enables you to stop a window from closing if the program user decides to cancel the close operation from a confirmation box.

Connect the *closeWidgetRequest* event as follows:

1. To the *self* attribute of a Variable part
2. To one of the *openModal* actions of a confirmation dialog
3. To the *suspendExecutionUntilRemoved* action of a confirmation dialog

The cancel event (for example, *clicked* of the cancel button) of the confirmation dialog should be connected to the *VAGen cancelCloseRequest:* action of the window to be closed. This connection requires a parameter to which you should connect the *self* attribute of the Variable part you assigned in the first connection above.

VAGen getFocusPart

The *VAGen getFocusPart* action enables you to perform standard CUA operations, such as Cut, Copy, Paste, and Clear from a menu.

You use this action by connecting some event (for example, *clicked* of the Cut menu button) to the *VAGen getFocusPart* action of a Window part. The result of this connection should be connected to *self* of a Variable part.

You should also connect the menu button to an unlisted feature of the variable that corresponds to the action that the *VAGen getFocusPart* action should perform. For example, *cutSelection*, *copySelection*, *paste*, or *clearSelection*.

VisualAge Generator extensions to VisualAge Smalltalk data types

VisualAge Generator provides some extensions to the VisualAge Smalltalk basic data types. It is recommended that these VAGen data types be used on parts that are connected to VAGen data items. This is especially true for the Boolean, Date and Time data types because of the implicit conversion that they perform on the data. The VisualAge Generator extensions are:

Boolean-VAGen
The Boolean-VAGen converter is like the Boolean converter except that on connection alignment, it can accept objects of types String and Integer in addition to objects of Boolean type. This allows VAGen data items of String and Integer types to be connected to attributes that expect Boolean data. See VisualAge Generator User's Guide for the rules used to convert String and Integer objects to Boolean objects.

Date-VAGen
The Date-VAGen converter is like the Date converter except that on connection alignment, it can accept objects of String and Integer types in addition to objects of Date type. This allows VAGen data items of String and Integer type to be connected to attributes that expect Date data. See VisualAge Generator User's Guide for the rules used to convert String and Integer objects to Date objects.

DBCS Only-VAGen
The DBCS Only-VAGen converter is like the DBCS Only converter except that it supports minimum and maximum values.

Number-VAGen
The Number-VAGen converter is like the Number converter except that it uses VisualAge Generator rules on rounding and truncation.

Time-VAGen
The Time-VAGen converter is like the Time converter except that on connection alignment, it can accept objects of String type in addition to objects of Time type. This allows VAGen data items of String type to be connected to attributes that expect Time data. See VisualAge Generator User's Guide for the rules used to convert String objects to Time objects.

Note: VisualAge Smalltalk recommends that you use **Properties Table** in place of **Notebook Style** settings view. **Properties Table** is used as the default unless the VisualAge Notebook Style settings views feature is loaded and **Notebook Style** is selected as the preferred settings view from the VisualAge Preferences window. If **Notebook Style** is selected,

you will need to load the configuration map **VAGen GUI Settings** to be able to access the settings view for these VAGen data types. Otherwise, the **Customize** button from the part's settings view is disabled when one of these data types is selected.

Graphical user interfaces in Java

The VisualAge for Java product ships a parts palette that includes templates for creating many visual and nonvisual program parts. VisualAge Generator ships the following extensions to the VisualAge for Java parts palette:

- An additional VisualAge Generator category and parts

All of the VisualAge Generator extensions to the parts palette have names that begin with the **VAGen** prefix. The basic parts palette shipped with VisualAge for Java is described in the VisualAge for Java online help.

For more information on visual programming and the visual parts of a GUI program, refer to the VisualAge for Java task information in the online help. For information on defining nonvisual parts, refer to the other chapters in this book.

VisualAge Generator parts category for Java

The VisualAge Generator product ships categories and parts that are added to the VisualAge for Java parts palette during installation. The following table shows the VisualAge Generator categories and their parts.

- VAGen Record
- VAGen Table
- VAGen Program
- VAGen Function

VAGenRecordPart

Select the **VAGenRecordPart** part to add a VisualAge Generator record to the free-form surface.

VAGenRecordPart properties:

byteData
The *byteData* property represents the contents of the record as a byte array. Use the*byteData* property if you wish to share or pass the record part's contents. The *byteData* property is readable, writable, and bound.

data The *data* property represents the contents of the record as a string. Use *data* property to display the record part's contents as a string. Due to internal conversions involved with the data property, if you wish to

share or pass the record part's contents, use *byteData* property instead to maintain its integrity. The *data* property is readable, writable, and bound.

this The *this* property represents the part itself. Connecting to this property of a record part to pass parameter to a function part expecting a record or to pass a record part into another bean that contains a place holder for it (a VAGen variable).

The *this* property is readable.

data item properties and data item data properties
Two properties are created for each of the individual and top-level substructured data items in the record. One property represents the data item and the other property represents the data item data. These properties are bound properties. The properties that represent the data items are readable and bound. The properties that represent the data items data are readable, writable and bound.

VAGenRecordPart methods:

getByteData( )
The *getByteData( )* method returns the byte array that represents the contents of the record. This method is the get selector for the *byteData* property.

getValue( )
The *getValue( )* method returns the string that represents the contents of the record.

setByteData(byte[])
The *setByteData(byte[])* method sets the contents of the record to the byte array given. This method is the get selector for the *byteData* property.

setValue(java.lang.String)
The *setValue(java.lang.String)* method sets the contents of the record to the string given.

VAGenRecordPart events: The VAGenRecordPart does not have any real events. However, you can use any of the data item properties in the record as a source of an event-to-method connections. The event is the modification of the value of the data item when the application runs.

VAGenTablePart

Select the **VAGenTablePart** part to add VisualAge Generator tables to the free-form surface.

VAGenTablePart properties: Following is the list of properties of a VAGenTablePart, none of which can be set in a Properties dialog:

byteData
The *byteData* property represents the contents of the table as a byte array. Use the *byteData* property if you wish to share or pass the table part's contents. The *byteData* property is readable, writable, and bound.

data The *data* property represents the contents of the table as a string. Use data property to display the table part's contents as a string. Due to internal conversions involved with the data property, if you wish to share or pass the table part's contents, use *byteData* property instead to maintain its integrity. The *data* property is readable, writable, and bound.

table columns
The *table columns* property represents the data items that make up the columns of a table. The *table columns* property is readable and bound.

table columns data
The *table columns data* property represents the contents of the columns of a table as an array of strings. The *table columns data* property is readable, writable and bound.

this The *this* property represents the part itself.

Connecting to this property of a table part to pass parameter to a function part expecting a table or to pass a table part into another bean that contains a place holder for it (a VAGen variable). The *this* property is readable.

data item properties and data item data properties
Two properties are created for each of the individual and top-level substructured data items in the record. One property represents the data item and the other property represents the data item data. These properties are bound properties. The properties that represent the data items are readable and bound. The properties that represent the data items data are readable, writable and bound.

VAGenTablePart methods:

getByteData( )
The *getByteData( )* method returns the byte array that represents the contents of the table. This method is the get selector for the *byteData* property.

getValue( )
The *getValue( )* method returns the string that represents the contents of the table.

setByteData(byte[])
The *setByteData(byte[])* method sets the contents of the table to the byte array given. This method is the get selector for the *byteData* property.

setValue(java.lang.String)
The *setValue(java.lang.String)* method sets the contents of the table to the string given.

VAGenTablePart events: The VAGenTablePart does not have any real events. However, you can use any of the data item properties in the table as a source of an event-to-method connections. The event is the modification of the value of the data item when the application runs.

Additional VAGen parts used with data parts

VAGen Field part: This part represents a data item in a data part, and you can access its features by tearing off a data item property from a data part. It is not a part in the VAGen Parts category.

VAGen Field properties: Following is the list of properties of a VAGen Field, none of which can be set in a Properties dialog:

bigNumData
The *bigNumData* property represents the contents of the data item as a com.ibm.vgj.wgs.VGJBigNumber, which is a numeric value with decimal precision, if possible. The *bigNumData* property is readable, writable, and is an expert property.

data The *data* property represents the contents of the data item. The type of the data property depends on the type of the data item. The following table maps the data item type to the Java type that is contained by this property:

Table 1. Data item type compared to Java type

Data item type	Comparable Java type
any numeric value with no decimal precision	long
any numeric value with decimal precision	com.ibm.vgj.wgs.VGJBigNumber
Char	java.lang.String
DBCS	java.lang.String
Mixed	java.lang.String
Hex	byte[]

Use *data* property if you wish to share or pass the data item's contents. The *data* property is readable, writable, and bound.

intData
The *intData* property represents the contents of the data item as a Java int if possible. The *intData* property is readable, writable, and is an expert property.

longData
The *longData* property represents the contents of the data item as a Java long if possible. The *longData* property is readable, writable, and is an expert property.

shortData
The *shortData* property represents the contents of the data item as a Java short if possible. The *shortData* property is readable, writable, and is an expert property.

stringData
The *stringData* property represents the contents of the data item as a string. The stringDataproperty is readable, writable, and is an expert property.

this The *this* property represents the data item itself. Connecting to this property of a data item to pass parameter to a function part expecting a data item. The *this* property is readable.

value The *value* property represents the contents of the data item as a Java Object. The following table maps the data item type to the Java object type that is contained by this property:

Table 2. Data item type compared to Java object type

Data item type	Comparable Java object type
any numeric value with no decimal precision	java.lang.Long
any numeric value with decimal precision	com.ibm.vgj.wgs.VGJBigNumber
Char	java.lang.String
DBCS	java.lang.String
Mixed	java.lang.String
Hex	byte[]

The *value* property is readable, writable, and is an expert property.

data item properties and data item data properties
If a VAGen Field represents a substructured data item, two properties are created for each of the individual and top-level substructured data items in the record. One property represents the data item and the

other property represents the data item data. These properties are bound properties. The properties that represent the data items are readable and bound. The properties that represent the data items data are readable, writable and bound.

VAGen Field methods:

getBigNumData( )
The *getBigNumData( )* method returns the contents of the data item as a com.ibm.vgj.wgs.VGJBigNumber. If the contents of the data item can not be converted to a numeric value with decimal precision, an exception will occur. This method is the get selector for the *bigNumData* property.

getByteData( )
The *getByteData( )* method returns the byte array that represents the contents of the data item. This method is the get selector for the *byteData* property.

getIntData( )
The *getIntData( )* method returns the contents of the data item as a Java int. If the contents of the data item can not be converted to an integer, an exception will occur. This method is the get selector for the *intData* property.

getLongData( )
The *getLongData( )* method returns the contents of the data item as a Java long. If the contents of the data item can not be converted to a long, an exception will occur. This method is the get selector for the *longData* property.

getShortData( )
The *getShortData( )* method returns the contents of the data item as a Java short. If the contents of the data item can not be converted to a short, an exception will occur. This method is the get selector for the *shortData* property.

getStringData( )
The *getStringData( )* method returns the contents the contents of the data item as a string. This method is the get selector for the *stringData* property.

getValue( )
The *getValue( )* method returns the Java object that represents the contents of the data item. The following table maps the data item type to the Java object type that is returned by this method:

Table 3. Data item type compared to Java object type

Data item type	Comparable Java object type
any numeric value with no decimal precision	java.lang.Long
any numeric value with decimal precision	com.ibm.vgj.wgs.VGJBigNumber
Char	java.lang.String
DBCS	java.lang.String
Mixed	java.lang.String
Hex	byte[]

setBigNumData(com.ibm.vgj.wgs.VGJBigNumber)
The *setBigNumData(com.ibm.vgj.wgs.VGJBigNumber)* method sets the contents of the data item to the com.ibm.vgj.wgs.VGJBigNumber given. If the value given cannot be converted to this data item's type, an exception will occur. This method is the set selector for the *bigNumData* property.

setByteData(byte[])
The *setByteData(byte[])* method sets the contents of the data item to the byte array given. This method is the get selector for the *byteData* property.

setIntData(int)
The *setIntData(int)* method sets the contents of the data item to the integer given. If the value given cannot be converted to this data item's type, an exception will occur. This method is the set selector for the *intData* property.

setLongData( )
The *setLongData( )* method sets the contents of the data item to the long given. If the value given cannot be converted to this data item's type, an exception will occur. This method is the set selector for the *longData* property.

setShortData( )
The *setShortData( )* method sets the contents of the data item to the short given. If the value given cannot be converted to this data item's type, an exception will occur. This method is the set selector for the *shortData* property.

setStringData(java.lang.String)
The *setStringData(java.lang.String)* method sets the contents of the data item to the string given. If the value given cannot be converted to this data item's type, an exception will occur. This method is the set selector for the *stringData* property.

setValue(java.lang.Object)
The *setValue(java.lang.String)* method sets the contents of the data item to the Java object given. If the value given can not be converted to this data item's type, an exception will occur. This method is the set selector for the *value* property.

setValueToDefault( )
The *setValueToDefault( )* method sets the contents of the data item to its default value (zero for the numeric data items and blanks for character data items).

VAGen Field events: The VAGen Field does not have any real events. However, if it represents a substructured data item, you can use any of the data item properties in it as a source of an event-to-method connection. The event is the modification of the value of the data item when the application runs.

VAGen Array Field: This part represents an occurs data item in a data part, and you can access its features by tearing off an occurs data item property from a data part. It is not a part in the VAGen Parts category.

Note: VAGen Array Field parts are indexed from 0 and VAGen Array Field methods operate on this assumption. For example, the integers in the method *getFieldsInRange(int, int)* are zero-based.

VAGen Array Field properties: Following is the list of properties of a VAGen Array Field, none of which can be set in a Properties dialog:

bigNumArrayData
The *bigNumArrayData* property represents the contents of the data item as an array of com.ibm.vgj.wgs.VGJBigNumber(s) if possible. The *bigNumArrayData* property is readable, writable, and is an expert property.

byteArrayData
The *byteArrayData* property represents the contents of the data item as a two-dimensional array of bytes. The *byteArrayData* property is readable, writable, and is an expert property.

data The *data* property represents the contents of the data item, which is an array of values. The type of the values depends on the type of the data item. The following table maps the data item type to the Java type that is contained in this array:

Table 4. Data item type compared to Java type

Data item type	Comparable Java type
any numeric value with no decimal precision	long

Table 4. Data item type compared to Java type (continued)

Data item type	Comparable Java type
any numeric value with decimal precision	com.ibm.vgj.wgs.VGJBigNumber
Char	java.lang.String
DBCS	java.lang.String
Mixed	java.lang.String
Hex	byte[]

Use *data* property if you wish to share or pass the data item's contents. The *data* property is readable, writable, and bound.

intArrayData
The *intArrayData* property represents the contents of the data item as an array of Java int(s) if possible. The *intArrayData* property is readable, writable, and is an expert property.

longArrayData
The *longArrayData* property represents the contents of the data item as an array of Java long(s) if possible. The *longArrayData* property is readable, writable, and is an expert property.

shortArrayData
The *shortArrayData* property represents the contents of the data item as an array of Java short(s) if possible. The *shortArrayData* property is readable, writable, and is an expert property.

stringArrayData
The *stringArrayData* property represents the contents of the data item as an array of string. The *stringArrayData* property is readable, writable, and is an expert property.

this The *this* property represents the data item itself. Connecting to this property of a data item to pass parameter to a function part expecting a data item or to display the data item contents in a JTable. The *this* property is readable.

data item properties and data item data properties
If a VAGen Array Field represents a substructured occurs data item, two properties are created for each of the data items in the substructure: one property represents the data item and the other property represents the data item data. The properties that represent the data items are readable and bound. The properties that represent the data items data are readable, writable and bound.

VAGen Array Field methods:

addNewRowAfter(int)

The *addNewRowAfter(int)* method adds an empty row after the given row number. The row number is 0-based. If this occurs item is connected to a JTable, the empty row will now show up in the JTable allowing the end-user to fill in the columns information.

addNewRowBefore(int)

The *addNewRowBefore(int)* method adds an empty row before the given row number. The row number is 0-based. If this occurs item is connected to a JTable, the empty row will now show up in the JTable allowing the end-user to fill in the columns information.

addNewRowLast()

The *addNewRowLast()* method adds an empty row at the end of the valid rows. If this occurs item is connected to a JTable, the empty row will now show up at the end of the JTable allowing the end-user to fill in the columns information.

getBigNumArrayData( )

The *getBigNumArrayData( )* method returns the contents of the data item as an array of com.ibm.vgj.wgs.VGJBigNumber(s). If the contents of the data item can not be converted to an array of numeric value with decimal precision, an exception will occur. This method is the get selector for the *bigNumArrayData* property.

getByteArrayData( )

The *getByteArrayData( )* method returns a two-dimensional array of bytes that represents the contents of the data item. This method is the get selector for the *byteArrayData* property.

getColumnClass(int)

The *getColumnClass(int)* method is part of the Swing's TableModel interface. It returns java.lang.Object class for all of the data items in its substructure.

getColumnCount()

The *getColumnCount()* method is part of the Swing's TableModel interface. It returns the number of data items in the occurs data item's substructure.

getColumnName(int)

The *getColumnName(int)* method is part of the Swing's TableModel interface. It returns the name of the data item positioned at the given column number in the occurs data item's substructure.

getElementAt(int)

The *getElementAt(int)* method is part of the Swing's ListModel interface. It returns the value of the element at the specified index.

getFieldAt(int)

The *getFieldAt(int)* method returns the element at the specified index of the occurs data item. You can use this method to pass the element at the specified index of the array as a parameter to the VAGenProgramPart.

getFieldsInRange(int, int)

The *getFieldsInRange(int, int)* method returns elements between two indexes of an occurs item. You can connect the result of this method to the model property of a JTable to get only a certain range of the occurs item.

getIntArrayData( )

The *getIntArrayData( )* method returns the contents of the data item as an array of Java int(s). If the contents of the data item cannot be converted to an array of integers, an exception will occur. This method is the get selector for the *intArrayData* property.

getLongArrayData( )

The *getLongArrayData( )* method returns the contents of the data item as an array of Java long(s). If the contents of the data item cannot be converted to an array of longs, an exception will occur. This method is the get selector for the *longArrayData* property.

getRowCount()

The *getRowCount()* method is part of the Swing's TableModel interface. It returns the count of elements in the occurs data item with non-default values.

getSelectedItem()

The *getSelectedItem()* method is part of the Swing's ComboBoxModel interface. It returns the value of the selected item from the JComboBox if the occurs data item is connected to a JComboBox's model.

getShortArrayData( )

The *getShortArrayData( )* method returns the contents of the data item as an array of Java short(s). If the contents of the data item cannot be converted to an array of shorts, an exception will occur. This method is the get selector for the *shortArrayData* property.

getSize()

The *getSize()* method is part of the Swing's ListModel interface. It returns the count of elements in the occurs data item with non-default values.

getStringArrayData( )

The *getStringArrayData( )* method returns the contents of the data item as an array of strings. This method is the get selector for the *stringArrayData* property.

getValue( )

The *getValue( )* method returns the value of the first element in the Array (index = 0). The following table maps the data item type to the Java object type that is returned by this method:

Table 5. Data item type compared to Java object type

Data item type	Comparable Java object type
any numeric value with no decimal precision	java.lang.Long
any numeric value with decimal precision	com.ibm.vgj.wgs.VGJBigNumber
Char	java.lang.String
DBCS	java.lang.String
Mixed	java.lang.String
Hex	byte[]

getValueAt(int)

The *getValueAt(int)* method returns the value of the element at the specified index of the occurs data item. You can connect the result of this method to anything you would normally connect to the data property of a non-occurs item such as JTextField's text .

getValueAt(int, int)

The *getValueAt(int, int)* method is part of the Swing's TableModel interface. It returns the value of cell <row, column> (row is the first integer given, column is the second integer given. Cell <row, column> is the occur row of the data item at position column in the occurs data item's substructure.

getValuesInRange(int, int)

The *getValuesInRange(int, int)* method returns the values of elements between two indexes of an occurs item.

isCellEditable(int, int)

The *isCellEditable(int, int)* method is part of the Swing's TableModel interface. It returns true for all cells <row, column>.

removeRowAt(int)

The *removeRowAt(int)* method removes the row at the specified index. The row index is 0-based. If this occurs item is connected to a JTable, the row will now be removed from the JTable.

setBigNumArrayData(com.ibm.vgj.wgs.VGJBigNumber[])

The *setBigNumArrayData(com.ibm.vgj.wgs.VGJBigNumber)* method sets the contents of the occurs data item to the array of com.ibm.vgj.wgs.VGJBigNumber(s) given. If the value given can not

be converted to this data item's type, an exception will occur. This method is the set selector for the *bigNumArrayData* property.

setByteArrayData(byte[])
The *setByteArrayData(byte[])* method sets the contents of the data item to the two-dimensional array of bytes given. This method is the get selector for the *byteArrayData* property.

setIntArrayData(int[])
The *setIntArrayData(int)* method sets the contents of the data item to the array of integers given. If the value given can not be converted to this data item's type, an exception will occur. This method is the set selector for the *intArrayData* property.

setLongArrayData([])
The *setLongArrayData([])* method sets the contents of the data item to the array of longs given. If the value given cannot be converted to this data item's type, an exception will occur. This method is the set selector for the *longArrayData* property.

setSelectedItem(java.lang.Object)
The *setSelectedItem()* method is part of the Swing's ComboBoxModel interface. It saves the value of the selected item from the JComboBox if the occurs data item is connected to a JComboBox's model.

setShortArrayData([])
The *setShortArrayData([])* method sets the contents of the data item to the array of shorts given. If the value given can not be converted to this data item's type, an exception will occur. This method is the set selector for the *shortArrayData* property.

setStringArrayData(java.lang.String[])
The *setStringData(java.lang.String[])* method sets the contents of the data item to the array of strings given. If the value given can not be converted to this data item's type, an exception will occur. This method is the set selector for the *stringArrayData* property.

setValue(java.lang.Object)
The *setValue(java.lang.String)* method sets the contents of the first element of the Array to the Java object given. If the value given can not be converted to this data item's type, an exception will occur. This method is the set selector for the *value* property.

setValueAt(int, java.lang.Object)
The *setValueAt(int, java.lang.Object)* method replaces the value of the element at the specified index with the Java object given. If the value given can not be converted to this data item's type, an exception will occur.

setValueAt(java.lang.Object, int, int)

The *setValueAt(int, int)* method is part of the Swing's TableModel interface. It replaces the value of the cell <row, column> with the object given (row is the first integer given, column is the second integer given. Cell <row, column> is the occur row of the data item at position column in the occurs data item's substructure.

setValuesInRange(int, int, java.lang.Object[])

The *setValuesInRange(int, int)* method sets the values of elements between two indexes of an occurs item to the array of values given.

setValueToDefault( )

The *setValueToDefault( )* method sets the contents of the first element of the Array to its default value (zero for the numeric data items and blanks for character data items).

VAGen Array Field events: The VAGen Array Field does not have any real events. However, if it represents a substructured data item, you can use any of the data item properties in it as a source of an event-to-method connection. The event is the modification of the value of the data item when the application runs. VAGen Logic Parts include VisualAge Generator logic parts you can use in building a GUI program. These parts are nonvisual. The parts are described in the following sections.

VAGenProgramPart

Select the **VAGenProgramPart** part to add a VisualAge Generator program or a non-VisualAge Generator program to the free-form surface. You can use the VAGen Program part as a way to visually call server programs.

VAGenProgramPart properties:

lastResult

The *lastResult* property represents an object of the type HptProgramResult. The *lastResult* property is the result of the last call to this VAGen Program.

The HptProgramResult object type is defined with the following features:

Properties

error The *error* property returns a boolean indicating if the last invocation of this program resulted in a non-zero return code. The error property is readable only.

errorObject

The *errorObject* property represents an object of the type HptProgramError if the last invocation of this program resulted in a non-zero return code. Otherwise, this property will return null.

The HptProgramError object type is defined with the following features:

dateAndTime
The *dateAndTime* property returns a java.lang.String. The *dateAndTime* property is readable and writable.

errorText
The *errorText* property returns a java.lang.String. The *errorText* property is readable and writable.

errorTextReplace
The *errorTextReplace* property returns a java.lang.String. The *errorTextReplace* property is readable and writable.

errorTextReplace
The *errorTextReplace* property returns a java.lang.String. The *errorTextReplace* property is readable and writable.

locus The *locus* property returns a java.lang.String. The *locus* property is readable and writable.

origin The *origin* property returns a java.lang.String. The *origin* property is readable and writable.

reasonCode
The *reasonCode* property returns an int. The *reasonCode* property is readable and writable.

returnCode
The *returnCode* property returns an int. The *returnCode* property is readable and writable.

errorText
The *errorText* property returns a java.lang.String containing the error text from the last invocation of this program, if an error was produced. Otherwise, this property will return null. The *errorText* property is readable and writable.

returnCode
The *returnCode* property returns an int. containing the return code from the last invocation of this program. The *returnCode* property is readable and writable.

The *lastResult* property is readable and bound.

linkageInfo
The *linkageInfo* property represents an object of the type

HptCallLinkageInformation, and contains necessary linkage information for making a call to another function or program.

The HptCallLinkageInformation class is defined with the following properties:

appType

The *appType* property specifies the remote application type. *appType* can have one of the following values:

VG The called program is a generated VisualAge Generator application. An additional parameter is automatically passed to the server to allow the server to return an error code to the middleware if the server application ends abnormally.

NON_VG

The called program was developed using a tool other than VisualAge Generator. Only the parameters passed on the call are passed to the called program.

conversionTable

The *conversionTable* property specifies the name of the conversion table used to perform automatic data conversion on the call to the remote application. The name is a 9-byte character array containing a null-terminated character string. Some names have a special meaning:

***** Conversion is performed on the client using the default conversion table. You must enclose the asterisk in single quotes.

On OS/2, AIX, and Windows systems, the default is the conversion table specified in environment variable EZERCVT. If EZERCVT is not specified, the default is conversion table ELACN*xxx* (OS/2 or AIX) or ELACW*xxx* (Windows), where *xxx* is the national language code specified in environment variable EZERNLS. If EZERNLS is not specified, the default national language code is ENU.

BINARY

Only binary fields are converted. The byte order in the binary field is reversed.

This table is used with OS/2 and Windows clients communicating with AIX servers, and vice versa, when both the client and the server are running under the same code page.

NONE
No conversion is performed.

externalName
The *externalName* property specifies the name of the entry point in the DLL named in the library. The *externalName* value is ignored if *isRemote* is **true**.

isRemote
The *isRemote* property is a Boolean value that specifies if a call is to a remote or local function.

is32Bit
The *is32Bit* property is a Boolean value that tells whether a called DLL function is a 32 Bit or 16 Bit function. The *is32Bit* property is ignored if *isRemote* is **true**.

library
If *isRemote* is **true**, the *library* property specifies the name of the library that contains function to be called.

If *externalName* and *programName* are empty, *library* will also be the function name.

If *isRemote* is **true**, the *library* property specifies the OS/400 program library name. The name is a 20-byte character array containing a null-terminated character string. This value is used only with the Client Access/400 and Java400 protocols. It specifies the name of the OS/400 library that contains the called program. The default value is the application name if the array contains a null string.

linkageTableName
The *linkageTableName* property specifies the file name of the linkage table to be used if run-time bind is specified for the Protocol parameter.

If not specified (null string), the linkage table file name is obtained from environment variable CSOLINKTBL.

If the name is not fully qualified, the VisualAge Generator middleware uses the current DPATH (for OS/2) or PATH (for Windows) search path to find the table.

location
The *location* property specifies the protocol-dependent server system name. The name is a 20–byte character array containing a null-terminated character string.

The following table shows the meaning of the identifier by protocol and the default value if a name is not specified (null

string).

Protocol	Meaning of location	Default value
CICS DPL	CICS system identifier	System identifier defined for **applname** in the CICS tables.
CICSCLIENT	CICS system identifier	First system identifier specified in the CICS client initialization file.
DCE, DCESECURE, DCECICS, DCEIMS, DCEVM	Location where the server advertises in the DCE CDS database. The location is specified in the configuration file used when the VisualAge Generator DCE server program is started.	No default.
APPCIMS	CPIC side information identifier. The side information specifies: • Partner LU Alias • Transaction Program Name • Mode Name	No default
VG	See VisualAge Generator routing table description	Host defined for **applname in routing table**
TCPIP	TCP/IP hostname	No default
NPIPE	For remote NPIPE support (IBM's LAN Server), specify the COMPUTERNAME value from the LAN server's IBMLAN.INI file. For local NPIPE support, specify LOCAL.	No default
CA/400	AS/400 system identifier	The managing OS/400 system
Java400	AS/400 system identifier	No default

luwType

The *luwType* property specifies the logical unit of work type. Values are:

CLIENT

Unit of work is under client control.

Server updates are not committed or rolled back until the client requests commit or rollback using the EZECOMIT or EZEROLLB services of VisualAge Generator or the *commit* or *rollback* actions of the VisualAge Generator *commSession* property for the

class on whose free-form surface this part was dropped. Server applications cannot request commit or rollback.

Environments which do not support client-controlled unit of work will ignore this value.

SERVER
Server unit of work is independent of the client's unit of work. Commit (or rollback on abnormal termination) is automatically issued when the server returns. Server applications can request rollback.

parmform
The *parmform* property specifies the parameter format.

This option is supported only when calling through the CICS OS/2 ECI or CICS Client ECI. It is ignored for all other types of middleware.

Possible values for *parmform* are:

COMMPTR
The server program expects to be called using the CSP/AE parameter-passing convention that uses pointers in the COMMAREA. Use only with MVS CICS or VSE CICS server programs that were generated or coded to use this parameter-passing convention.

COMMDATA
The server program expects to receive the parameter values in the CICS COMMAREA. The parameter values passed on the call are moved into a single buffer, each value adjoining the previous value without regard for boundary alignment. On return from the remote call, the values returned in the output buffer are moved back to the corresponding parameters that were passed on the call.

programName
The *programName* property specifies the name of the server program that is being called.

The name is a null-terminated character string with a maximum length of eight characters plus the null terminator.

protocol
The *protocol* property specifies the communications protocol used to communicate with the client application.

Valid values are:

Runtime Bind
The communications protocol is read from the linkage table at run time. In addition, the following option values are read from the linkage table and any corresponding option specified in the *linkageInfo* settings is ignored:
luwType
appType
parmform
conversionTable
location
serverId
library

CICS Client ECI
CICS Client External Call Interface

Client Access/400
Client Access/400

Java400
Java driver to connect to AS/400 system

APPC to IMS
LU 6.2 connection to IMS message processing region

DCE RPC Secure
Distributed Computing Environment Remote Procedure Call (DCE RPC), no authorization checking

DCE RPC Secure
Distributed Computing Environment Remote Procedure Call (DCE RPC) with authorization checking

DCE to CICS
Distributed Computing Environment to CICS

DCE to IMS
Distributed Computing Environment to IMS message processing region

DCE to VM
Distributed Computing Environment to VM

LU2 Logical Unit 2

Name Pipes
Name Pipes

PACBASE
PACBASE

TCP/IP
Transmission Control Protocol/Internetwork Protocol

serverId
The *serverId* property specifies the protocol-dependent server channel or transaction name. The name is a 20–byte character array containing a null-terminated character string.

The following table shows the meaning of the identifier by protocol and the default value if a name is not specified (null string).

Protocol	Meaning of Server Identifier
CICS, CICSCLIENT	Name of CICS transaction for the server. If client unit of work is specified, all applications called in the same unit of work must have the same server identifier. The default is the CICS server system mirror transaction.
DCE, DCESECURE	Server ID name advertised by the server in the DCE CDS database. The *serverId* is specified in the configuration file used when the VisualAge Generator DCE server program is started.

providerURL
This *providerURL* property specifies the host name and port of the name server used by the EJB client. The property value must have the following format: iiop://hostname:port, where hostname is the IP address or hostname of the machine on which the name server runs and port is the port number on which the name server listens.

this The *this* property represents the part itself.

The *this* property is readable.

VAGenProgramPart methods:

execute
The *execute* method runs the function or program. This method accepts parameters for the program or function on the connection.

Parameters can be added by selecting Build parameters from definition or Add parameter from the VAGen Program Part's pop-up menu.

executeDeferred:
The *executeDeferred:* method runs the function or program after the specified delay. This method accepts parameters for the program or function on the connection. The argument to this action is the delay

interval, which is specified in milliseconds. It is recommended that you do not use delays that are less than 100 milliseconds.

This action can be used to create a polling loop to wait on a certain resource to become available. The VAGen Logic part can check the resource and reschedule itself to executeDeferred again using a perform request structure. Once the resource is available, the loop can be terminated by simply not redispatching the VAGen Logic part again. Note that during the delay period, you are free to interact with the user interface and you can schedule other logic events to run prior to the timer expiration.

This action creates a background delay that will expire at the end of the delay interval. At that time, the logic part will be put on the execution queue to be processed at the next opportunity the queue is read. This means that there is no guarantee that the logic you have deferred will execute in the same order or at a set timer interval. The interval simply specifies when it would run at the earliest.

Parameters can be added by selecting Build parameters from definition or Add parameter from the VAGen Program Part's pop-up menu.

VAGenProgramPart events:

failure

The *failure* event signals that the call has failed. Both the new and the old value of the *lastResult* property is signalled with this event.

hasExecuted

The *hasExecuted* event signals that a function or program has been run.

success

The *success* event signals that the call was successful. Both the new and the old value of the *lastResult* property is signalled with this event.

VAGenFunctionPart

Select the **VAGenFunctionPart** part to add a VisualAge Generator function to the free-form surface.

VAGenFunctionPart properties:

returnValue

The *returnValue* property contains the value returned by the last invocation of this function part. The following table maps the return value type returned by the function to the Java object type that is contained by this property:

Table 6. Data item type compared to Java object type

Data item type	Comparable Java object type
any numeric value with no decimal precision	java.lang.Long
any numeric value with decimal precision	com.ibm.vgj.wgs.VGJBigNumber
Char	java.lang.String
DBCS	java.lang.String
Mixed	java.lang.String
Hex	byte[]

The *returnValue* property is readable and bound.

this The *this* property represents the part itself.

The *this* property is readable.

VAGenFunctionPart methods: The following methods are associated with VAGenFunctionPart:

execute

The *execute* method runs the function or program. This method accepts parameters for the program or function on the connection.

Parameters can be added by selecting Build parameters from definition or Add parameter from the VAGen Program Part's pop-up menu.

executeDeferred:

The *executeDeferred:* method runs the function or program after the specified delay. This method accepts parameters for the program or function on the connection. The argument to this action is the delay interval, which is specified in milliseconds. It is recommended that you do not use delays that are less than 100 milliseconds.

This action can be used to create a polling loop to wait on a certain resource to become available. The VAGen Logic part can check the resource and reschedule itself to executeDeferred again using a perform request structure. Once the resource is available, the loop can be terminated by simply not redispatching the VAGen Logic part again. Note that during the delay period, you are free to interact with the user interface and you can schedule other logic events to run prior to the timer expiration.

This action creates a background delay that will expire at the end of the delay interval. At that time, the logic part will be put on the execution queue to be processed at the next opportunity the queue is

read. This means that there is no guarantee that the logic you have deferred will execute in the same order or at a set timer interval. The interval simply specifies when it would run at the earliest.

Parameters can be added by selecting Build parameters from definition or Add parameter from the VAGen Program Part's pop-up menu.

getBigNumReturnValue()
The *getBigNumReturnValue()* method returns the *returnValue* property converted the type com.ibm.vgj.wgs.VGJBigNumber

getByteNumReturnValue()
The *getByteNumReturnValue()* method returns the *returnValue* property converted the type byte[].

getIntReturnValue()
The *getIntReturnValue()* method returns the *returnValue* property converted the type int

getLongReturnValue()
The *getLongReturnValue()* method returns the *returnValue* property converted the type long

getShortReturnValue()
The *getShortReturnValue()* method returns the *returnValue* property converted the type short

getStringReturnValue()
The *getStringReturnValue()* method returns the *returnValue* property converted the type java.lang.String

VAGenFunctionPart events: The following event is associated with VAGenFunctionPart:

hasExecuted
The *hasExecuted* event signals that a function or program has been run.

VAGenVariable Part
Select the VAGenVariable part and drop it on the free form surface to enable your program to work with a part that is created at run time. A variable is a placeholder for the actual part, much like a parameter in an ordinary programming language.

VAGenCommSession Part
Select the VAGenCommSession Part to add a communication session to the free form surface. Use this part to explicitly define which communication session your other VAGen parts use.

VAGenCommSession attributes: Two attributes associated with the VAGen Communications Session part are *VAGen inheritsCommSession* and *VAGen commSessionOwner*.

VAGen inheritsCommSession is a boolean attribute that controls how to look for the instance of a VAGen Communications Session part when a call to a VAGen server is being issued. If set to true, the *VAGen commSessionOwner* is checked first. If the *VAGen commSessionOwner* is set to nil, then the parent is checked. If the *VAGen inheritsCommSession* attribute is false, then the current part will return the session that is stored in *VAGen commSession*.

VAGen commSessionOwner is an attribute that can be set to any instance of a subclass of *AbtAppBldrNonVisual*. *VAGen commSessionOwner* is used in a hierarchy of parts to control which parts actually have the *VAGen commSession* attribute set to an instance of a VAGen Communications Session part. If *VAGen commSessionOwner* is set, and *VAGen inheritsCommSession* is set to true, the session object will be looked for in that part.

VAGenCommSession properties: The following properties are associated with the VAGen Communications Session part:

password
Specifies the password to be used for this communications session. This property is used in combination with the userID property. The value needs to be available when the Java GUI requires communication to an AS/400 server program through the Java400 protocol.

userID
Specifies the user ID to be used for this communications session. This property is used in combination with the password property. The value needs to be available when the Java GUI requires communication to AS/400 server program through the Java400 protocol.

Additional VisualAge Generator Features for VisualAge Java Beans

VAGenCommSession

The *VAGenCommSession* property represents an object of the type HptCommSession. This is the same type that is used when a VAGenCommSession Part is dropped on the free-form surface.

When the first VisualAge Generator part is added to a Java bean, VAGen adds the *VAGenCommSession* property to the Java bean's interface.

The *VAGenCommSession* property allows users to share the communication session created to handle server calls between multiple beans. Connecting a

VAGenCommSession Part (HptCommSession) from another bean to this property allows the beans to share the same communication session.

If this property is not set, a communication session is created by default when needed.

Chapter 2. Programs

A program refers to any of the following types of VisualAge Generator programs:

- Programs that communicate with the user via a text (3270 or character based) user interface.
- Stand alone batch programs.
- Called programs (server programs or local sub-programs).

In short, a program is any VisualAge Generator program other than a GUI program.

When you create a program, you must define general information such as, the type of program you are creating, the map group the program will use, the working storage name, the help map group, the message table prefix, and the first map name. You also define information about function keys and implicits. Finally, you associate logic and data parts with the program using the main function list, the table and additional records list, and the called parameter list.

Program elements

Table 7. Program elements

Element	COBOL											C++								TEST FACILITY
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
Allow implicits	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Bypass edit keys	x		x	c		c		x		x	x	x	x	x	x	x	x	x	x	x
Called parameter list	x	x	x	x	x	c	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Table 7. Program elements (continued)

Element	COBOL											C++								TEST FACILITY
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
Execution mode	c		c	c		c		c		c	i	c	c	c	c	c	c	c	c	x
F1-12 = F13-24	x		x	x		x		x		x	x	x	x	x	x	x	x	x	x	x
First map	x		x	x		c		x		x	x	x	x	x	x	x	x	x	x	x
First UI			c			c		c				c	c	c	c	c	c	c	c	c
Flow statements	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Help key	x		x	c		c		x		x	x	x	x	x	x	x	x	x	x	x
Help map group name	x		x	x		x		x		x	x	x	x	x	x	x	x	x	x	x
Keep after use	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
Main function list	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Map group name	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Message table prefix	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Program name	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Program type	x	c	x	x	c	c	c	x	c	x	x	x	x	x	x	x	x	x	x	x
Prologue	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
PSB name			c	c	c	c	c	c	c	i	i	i	i	i	c	c	c	i	c	x
Structure list	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Table and additional records list	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Table 7. Program elements (continued)

Element	COBOL											C++								TEST FACILITY
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
Working storage	x	x	x	x	x	c	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Legend: In this table, the following characters are used to indicate the level of support:

x	Supported
c	Supported with compatibility considerations
i	Ignored.
v	Supported with VisualAge Generator generation, but not with TeamConnection build
*	Resource association file referenced only at runtime
blank	Not supported

Allow implicits

Allow implicits enables you to have the VisualAge Generator test and generation facilities create implicit data item definitions.

Uses

Implicit data item definitions are needed for unqualified data item names referenced in the program that are not defined in any of the records, tables, or maps used in the program.

If you do not specify Allow implicits, the test and generation facilities bypass all processing involved in creating implicit data items. These facilities issue error messages for any undefined data item.

Performance information for Allow implicits

Test and generation performance improve when you do not Allow implicits.

Target environments for Allow implicits

Supported in all environments without compatibility considerations.

Bypass edit keys

You can specify up to five function keys for the program user to use to bypass map edits and map edit routines.

Bypass edit keys

Uses

When the program user presses a bypass edit key, data is not passed to the program, and the program continues processing at the statement following the terminal I/O function (either the first map or a CONVERSE statement).

The data on a map when the program user presses a bypass edit key is not saved.

PA keys are treated as bypass edit keys in a generated program.

The values specified during program specification are the default keys for the maps used by the program. However, the values you assign for the bypass edit keys during map definition override the values specified during program specification.

For example, if three keys are specified during program specification, but only one is specified during map definition, only the key specified during map definition can be used for that map.

Note: You cannot use one function key as both a bypass edit key and the help key.

Target environments for bypass edit keys

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	PF6 is reserved for a panel recovery function in this environment. If you press PF6, it is treated as pressing the Clear key. The PF key value is not passed back to the program. Avoid using PF6 in this environment.
MVS batch	Not supported.
IMS/VS	IMS reserves the PA keys so they cannot be the default bypass edit keys. A specific PF key must be defined if the program user is allowed to bypass edits. If your installation uses PF12 for the IMS local copy function, you cannot use PF12 as a bypass edit key.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.

Environment	Compatibility Considerations
CICS for OS/2	None.
OS/400	None.
OS/2	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Called parameter list

The called parameter list shows the names and types of parameters your program receives.

Uses

The following can be specified for each parameter to be passed to this program:

Name The name of the data item, record, or map received as a parameter.

Type The part type of the parameter:
- Item
- Map
- Record

Definition considerations for called parameter list

The following restrictions apply to parameters:

- The maximum number of parameters is 30. Parameters can be maps, records, or data items. EZEDLPSB or EZEDLPCB can also be a parameter.

  Note: If you specify EZEDLPSB or EZEDLPCB as a parameter, specify Item as the type of parameter.
- The parameters must be listed in the same order as the arguments are listed in the CALL statement in the calling program.
- The number of parameters must equal the number of arguments.

- The parameter definitions must be 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.
- The data item must be defined using data item definition. A data item parameter cannot be a data item in a record, table, or map used by the called program.
- If the program is going to be a server program called from a remote system, the total length of all parameters must not exceed 32567 bytes.
- Parameters in the called parameter list cannot be used as I/O objects or specified as the working storage record for the called program.

Target environments for called parameter list

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	The preprocessor requires either EZEDLPSB or EZEDLPCB when generating for IMS/VS. Remote server programs in IMS/VS require EZEDLPSB but cannot accept EZEDLPCBs.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Execution mode

Execution mode defines the mode in which main transaction programs are started.

Uses

The following execution modes are valid for main transaction programs:

Nonsegmented
A CONVERSE does not mark the end of a unit of work. I/O locks and database and file positions are maintained across the CONVERSE.

Segmented
Each write to the terminal (CONVERSE or XFER with map or UI record) is the end of a unit of work.

Single segment
A single input from a terminal is processed and the program stops running after one of the following occurs:
- The program responds to the input.
- Control transfers to another program.

Programs running in single segment mode have the following limitations:
- Programs cannot use the CONVERSE I/O option.
- Programs must use the XFER statement with a map and a first map for terminal I/O operations.
- Programs with terminal I/O operations must use the XFER statement with a map to transfer to a program with a first map.

Definition considerations for segmented

On each write to the terminal, database and file changes are committed. The program saves the contents of records and maps across a CONVERSE, but not across an XFER with map or UI record.

The following information is also lost on the commit:
- I/O locks
- Database and file positions
- Main storage resources

Definition considerations for single segment

Each interaction with the program user at the terminal is the end of the unit of work. Database and file changes are committed and all resources are freed. The only data passed to the next segment is the data in the map variable and the data in the passed working storage record.

Target environments for execution mode

Environment	Compatibility Considerations
VM CMS	Segmented and single segment mode are simulated by the following: • Committing database changes prior to each CONVERSE • Refreshing single user table contents • Resetting to their default values the EZE words that are not saved across segments.
VM batch	Not supported.
CICS for MVS/ESA	The end of a segment is the equivalent of a CICS SYNCPOINT. All updates to recoverable files and databases are committed and all I/O locks and positions are freed. Nonsegmented programs run in conversational mode. Segmented programs run in pseudoconversational mode. For segmented programs, all main storage resources are free while the system waits for terminal input from the program user.
MVS/TSO	Same as VM CMS.
MVS batch	Not supported.
IMS/VS	Nonsegmented main transaction programs are not supported. EZESEGM is ignored. The end of a segment is a commit point. All updates to recoverable files and databases are committed and all I/O locks and positions are freed. For segmented programs, all main storage resources are free while the system waits for terminal input from the program user. IMS conversational processing is used for segmented programs if an IMS scratchpad area (SPA) length greater than 0 is specified as a COBOL generation option. IMS nonconversational processing is used for segmented programs if SPA length is equal to 0 or it is not specified.
IMS BMP	Not supported.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	Not supported.
CICS for OS/2	Same as CICS for MVS/ESA.
OS/400	Ignored, all programs operate as nonsegmented.
OS/2	Same as VM CMS.
AIX	Same as VM CMS.

Environment	Compatibility Considerations
HP-UX	Same as VM CMS.
CICS for AIX	Same as VM CMS.
Windows NT	Same as VM CMS.
CICS for Windows NT	Same as VM CMS.
Solaris	Same as VM CMS.
CICS for Solaris	Same as VM CMS.
Test Facility	None.

F1-12=F13-24

F1-12=F13-24 assigns the functions of the F1 to F12 function keys to the F13 to F24 function keys.

Uses

A program test to determine whether a single key, such as F1 has been pressed will test true if either F1 or F13 is pressed only if F1-F12=F13-F24 is specified.

Target environments for F1-12=F13-24

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2	None.
AIX	None.
HP-UX	None.

Environment	Compatibility Considerations
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

First map

First map specifies the name of a map to display when a main transaction first starts.

Uses

First map is an initial map on which the program user enters input before the first function is run.

Definition considerations for first map

The program enables the user to enter input from the first map before any other program logic runs. Any inputs are validated according to map edit specifications.

A program with a first map starts as the result of one of the following:
- An XFER statement
- A transfer without a map from a non-VisualAge Generator program
- The program being started by a user (entering a transaction code for CICS or IMS, running a CLIST for TSO, or running a runtime REXX exec for VM CMS)

If the program was started using an XFER with map, the program reads the map from the terminal prior to executing the first function. Otherwise, the program converses the map prior to executing the first function. In this case, the map fields are initialized as though a `SET map CLEAR` statement was performed.

When the map is read, the contents of the map are validated as specified by the map item edits. If the contents are not valid, the map displays with an error message. If the contents are valid, execution continues with the first function defined for the program.

First map cannot be specified for main batch, called batch, called transaction, or web transaction programs. You cannot use the DXFR statement to transfer control to a program that has a first map specified.

The first map must be part of the map group specified for the program.

Target environments for first map

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	If the program is transferred to or from another program using XFER with a map, the transferring and the transferred-to programs must share the same map group. A program with a first map can also be started with the IMS /FORMAT command, provided there is an IMS transaction code on the map. A program with a first map can also start with a deferred program-to-program message switch from a non-VisualAge Generator program. Refer to the IMS chapter in the *VisualAge Generator Design Guide* document for more information.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.

First map

Environment	Compatibility Considerations
CICS for Solaris	None.
Test Facility	None.

First UI record

First UI record is an initial UI record to which data is passed in a program link from one Web transaction program to another Web transaction program. The first UI record contains the definition of the data items that receive data.

Definition considerations for First UI record

The only valid record type that can be specified as a First UI record is a UI record. A First UI record can be specified only in a Web transaction program.

Target environments for First UI record

Environment	Compatibility Considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only for Web transaction programs.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only for Web transaction programs.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only for Web transaction programs.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2	Valid only for Web transaction programs.
AIX	Valid only for Web transaction programs.
HP-UX	Valid only for Web transaction programs.
CICS for AIX	Valid only for Web transaction programs.
Windows NT	Valid only for Web transaction programs.
CICS for Windows NT	Valid only for Web transaction programs.
Solaris	Valid only for Web transaction programs.
CICS for Solaris	Valid only for Web transaction programs.

Environment	Compatibility Considerations
Test Facility	Valid only for Web transaction programs.

Flow statements

Flow statements consist of the processing statements associated with each main (first-level) function in the program function list. Flow statements can identify the next main function to be executed.

Uses

Flow statements for a function are executed after the function is executed. If no flow statements are specified for the function, the default flow is to execute the next main function in the program function list.

Flow statements are described in "Chapter 10. Program processing statements" on page 363.

Flow statements are stored with the program definition and are not part of the function definition. A function can be used in more than one program with different flow definitions.

Target environments for flow statements

Supported in all environments without compatibility considerations.

Help key

If the program you are creating provides help information, help key specifies the Help function key for maps in this program.

Uses

F1 is the default help key, but you can use any function key from F1 to F24.

Note: You cannot have a function key be both a bypass edit key and the help key.

Definition considerations for help key

You can override the default help function key for individual maps during map definition.

Target environments for help key

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.

Environment	Compatibility Considerations
CICS for MVS/ESA	None.
MVS/TSO	F6 is reserved for a panel recovery function in this environment. F6 is treated as pressing the Clear key. The function key value is not passed back to the program. Avoid using F6 in this environment.
MVS batch	Not supported.
IMS/VS	If your installation uses F12 for the IMS local copy function, you cannot use F12 as a help function key.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Help map group name

Help map group name is the name of the map group containing the help maps you define for the program.

Uses

If your program does not provide help maps or if the help maps are in the same map group you specified for the Map group name, you do not need to specify the Help map group name.

Definition considerations for help map group name

Using a separate map group for your help maps is more efficient and can save storage during execution because the help maps are loaded only when necessary. You can specify only one help map group for each program.

Target environments for help map group name

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Keep after use

Specify Keep after use to have the use count for this table incremented when the program or program segment first references the table and decremented at the end of the program.

If you do not specify Keep after use, the use count is incremented at the first use of the table in a main function and decremented when the main function, program, or program segment ends.

Keep after use

Definition considerations for keep after use

VisualAge Generator Server for MVS, VSE, and VM and VisualAge Generator Server maintain a use count for all tables in use by a program.

When the use count goes to zero, the table contents are released from memory unless the table has been defined as Resident.

Target environments for keep after use

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	Not supported.

Main function list

The main function list shows the sequence and specifications of the main functions that can be executed as part of the program.

Uses

The function list determines the default sequence of execution. The default logic flow begins with reading and editing the First map or First UI record, if specified, and then executing each main function in turn, based on the order that they appear in the main function list.

Target environments for main function list

Supported in all environments without compatibility considerations.

Map group name

Map group name is the name of the map group that contains the maps used in your program.

Note: You must specify a map group name for all transaction programs except web transactions.

Uses

The map group contains the maps that you will use as one of the following:

- I/O objects
- First map
- Parameters in the called parameter list
- Arguments on a CALL or XFER with map statements

A called program might 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 must be used.

In addition, if a program transfers to another program using an XFER with a map, the transferring and transferred-to programs must use the same map group, or the same map must be defined in both map groups.

Performance information for map group name

For better performance, avoid sharing map groups between programs unless all maps are the same.

Target environments for map group name

Supported in all environments without compatibility considerations.

Message table prefix

Message table prefix is a 3 or 4 character prefix that identifies the message table for your program. The message table contains program messages.

Message table prefix

Uses

When the program runs, the national language support code for the environment where the program is running supplies a suffix to the table name prefix, forming the name of the message table. The following are the language codes:

Code	Language
CHS	Simplified Chinese
CHT	Traditional Chinese
DES	Swiss German
DEU	German
ENP	Uppercase English
ENU	US English
ESP	Spanish
FRA	French
ITA	Italian
JPN	Japanese
KOR	Korean
PTB	Brazilian Portuguese

Note: Uppercase English is not supported by AIX, OS/2, Windows NT, HP-UX, SCO OpenServer, and Solaris.

For example, if the table name prefix is PRX and the program was generated with Spanish as the runtime language, then the message table name would be PRXESP.

Definition considerations for message table prefix

The message table is accessed during test and execution when one of the following is true:

- EZEMNO is modified.
- An edit check fails for which you have specified an alternate edit error message.

When the program type is Web transaction:

- The table type does not have to be Message.
- The table must have 2 columns each of which is type CHA, MIX, or UNICODE.
- The table is only accessed at the Web Server by the UI Record Java Beans that were generated from UI Records which used the table.
- Access occurs when:
  - EZEUIERR sets a given item in error.
  - An edit check fails for which you have specified an alternate edit message

The given key is used on lookup.

Target environments for message table prefix

Supported in all environments without compatibility considerations.

Program name

Program name specifies the name of the program being defined, and also the name of the COBOL or C++ program generated for the program.

Definition considerations for program name

Naming conventions for programs:

- Maximum length is 7
- First character must be alphabetic (A-Z)
- Other characters can be alphanumeric (A-Z, 0-9)
- Cannot begin with the EZE prefix
- Cannot contain embedded blanks
- Cannot be a COBOL reserved word (in COBOL environments)
- Cannot be a DBCS name
- To avoid potential conflicts with the program names generated for the map groups, do not end the program name with FM or P1

Target environments for program name

Supported in all environments without compatibility considerations.

Program type

Program type indicates the method of processing used by a program.

Uses

You can specify one of the following types of programs:

Main transaction
You intend to start the program by a transfer from the system or another program.

The program user can interact with the program using maps.

Called transaction
You intend the program to be called from another program.

The program user can interact with the program using maps.

Parameters can be passed and reset by the called program.

Main batch
You intend to start the program by a transfer from the system or another program.

The program user cannot interact with the program using maps.

Called batch

You intend the program to be called from another program.

The program user cannot interact with the program using maps.

Parameters can be passed and reset by the called program.

Server programs called from remote clients must be specified as called batch programs.

Web transaction

You intend to start the program by a transfer from the system or another program.

The program user can interact with the program using HTML pages and forms.

The Segmented execution mode is implied.

Definition considerations for Main transaction and Main batch

You can start a main transaction program or a main batch program with a transfer from one of the following:

- the system
- a non-VisualAge Generator program
- a VisualAge Generator program

A block of working storage data can be passed to the program on transfer from a non-VisualAge Generator program or VisualAge Generator program. The block of storage is used to initialize the working storage record defined for the program.

Definition considerations for Web transaction

You can start a Web transaction program with a transfer from one of the following:

- the system
- a non-VisualAge Generator program
- a VisualAge Generator program

A First UI record can be defined and data can be passed to the First UI record on transfer from another Web transaction program.

There are no map groups in Web transaction programs.

Target environments for program type

Environment	Compatibility Considerations
VM CMS	Web transaction programs are not supported.
VM batch	Transaction programs are not supported.

Environment	Compatibility Considerations
CICS for MVS/ESA	None.
MVS/TSO	Web transaction programs are not supported.
MVS batch	Web transaction programs are not supported.
IMS/VS	Called transaction programs are not supported.
IMS BMP	Web transaction programs are not supported.
CICS for VSE/ESA	None.
VSE batch	Same as VM batch.
CICS for OS/2	Web transaction programs are not supported.
OS/400	Web transaction programs are not supported.
OS/2	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Prologue

The prologue area is unformatted text that describes the program.

Uses

The use of a prologue is optional; it is commentary only and does not affect program execution.

Target environments for prologue

Supported in all environments without compatibility considerations.

Program specification block (PSB) name

PSB name is the name of the PSB part that describes the IMS message queues and DL/I databases used in the program.

PSB name

Uses

The PSB definition is used in generating default DL/I call information.

Definition considerations for PSB name

The definition of the database PCBs in the IMS or DL/I PSB used with the program must match the definition of the PCBs in the PSB part, except for database names.

Target environments for PSB name

Environment	Compatibility Considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	The PSB name is the default DL/I PSB scheduled when the program is executed. The program can schedule another PSB instead of the default PSB by moving the alternate PSB name to special function word EZEDLPSB. The alternate PSB is scheduled the next time PSB scheduling is required. The program schedules the PSB on the first DL/I function following: • Program start • A commit or rollback
MVS/TSO	EZEDLPSB cannot be used to change the PSB name while the program is running. The DL/I PSB must be generated with CMPAT=YES specified on the PSBGEN macro.
MVS batch	Same as MVS/TSO.
IMS/VS	The PSB part used with a main transaction or main batch program cannot have the same name as the IMS PSB scheduled for the IMS transaction, even though the definitions of the PSB part and the IMS PSB must match. IMS assumes that the IMS PSB name is the same as the program name. To avoid confusion, choose a naming convention whereby the PSB part names can be derived in a consistent fashion from the corresponding program and IMS PSB name. EZEDLPSB cannot be used to change the PSB name while the program is running. The IMS PSB must be generated with CMPAT=YES specified on the PSBGEN macro.
IMS BMP	The PSB name is required; otherwise same as MVS/TSO.

Environment	Compatibility Considerations
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	EZEDLPSB cannot be used to change the PSB name while the program is running.
CICS for OS/2	Ignored.
OS/400	Ignored.
OS/2	Ignored.
AIX	Ignored.
HP-UX	Ignored.
CICS for AIX	The PSB name is ignored.
Windows NT	Same as CICS for AIX.
CICS for Windows NT	Same as CICS for AIX
Solaris	Ignored.
CICS for Solaris	The PSB name is ignored.
Test Facility	None.

Structure list

Structure list is a top-down structure of all functions in the program.

Uses

The first level in the program structure is defined by the First mapspecification and the program main function list. If First map is specified, the first entry in the first level shows FIRSTMAP as the I/O option and the First map as the I/O object. Otherwise, the first entry is the first main function in the program.

Lower levels in the structure are defined implicitly by coding language elements that invoke lower level functions. These language elements follow:

- FIND or TEST statement that invokes a function
- Function used as an error routine, which is invoked when an I/O error is returned on execution of the I/O option
- Function specified as a map edit routine for a variable field on the First map or a conversed map.

Target environments for structure list

Supported in all environments without compatibility considerations.

Table and additional record list

Table and additional records list specifies the tables and additional record definitions needed in the program.

Definition considerations for table and additional record list

In the table and additional records list, the following items must be specified:

- The names of all table parts referenced in the program, including tables specified as input edit routines for map variable fields. The list is used to verify references to tables by function invocation statements and to assure that the tables are available at program execution.

  Do not include the name of the message table in the table and additional records list. The name of the message table is included in the program by the message table prefix you specify during program specification.
- Record parts only if they are not specified as the program working storage record, as I/O objects, or in the called parameter list.

  Records in the list can be used as additional temporary storage. The program cannot reference level-77 data items in a working storage record when the record is included using the additional records list.
- Record redefinitions needed by the program.

For each table entry in the list you can also specify Keep after use. See "Keep after use" on page 71 for more information.

Target environments for table and additional record list

Supported in all environments without compatibility considerations.

Working storage

Working storage records define storage areas for temporary data items used in VAGen programs.

Definition considerations for working storage

Only one primary working storage record is named in the program specification. Use the table and additional record list to include additional working storage records.

The primary working storage record is initialized to blanks for CHA, DBCS, Unicode, and MIX, and zero for numeric data. For main programs, if a record is received from the transferring program, the primary working storage record is first initialized based on the type of data and then the received record is moved into the primary working storage record.

If the primary working storage record is longer than the received record, the extra data in the primary working storage record remains initialized based on its data type.

The structure of the received record must match the structure defined in the working storage record. Otherwise data that is not valid in the working storage record can cause abnormal termination of the program.

Target environments for working storage

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	If an input message to a main transaction consists of only the transaction name followed by blanks, the program assumes it starts with no working storage record being passed. The primary working storage record is initialized based on the type of data.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Chapter 3. Functions

A function is a logic block consisting of statements surrounding a central function, usually an I/O operation. The central function is defined by: the I/O option, the I/O object, and the I/O error routine. The I/O option can be a file or database access, a write and subsequent read of a text user interface map, or a write of a printer map.

Functions are included in a program by being named in a program main function list, named as a map item edit routine, named as an I/O error routine, named as a target of a TEST or FIND statement, dropped as a Function part on a GUI definition or invoked from within another function.

Statements that can be included in a function definition are described in Chapter 10. Program processing statements. The function elements are described in this section.

Function elements

Table 8. Function elements

Element	COBOL											GUI		C++								Java		
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	Windows NT	Test Facility
DL/I call			x	x	x	x	x	x	x														x
DL/I call - Database identifier			x	x	x	x	x	x	x														x
DL/I call - Scan for update			x	x	x	x	x	x	x														x
DL/I call - Scan in parent			x	x	x	x	x	x	x														x

Table 8. Function elements (continued)

Element	COBOL											GUI		C++								Java	Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	Windows NT	
DL/I call - Segment search arg			x	c	c	x	x	c	c														x
Function	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Function description	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x	x
Function local storage list	x	x	x	x	x	x	x	x	x	x	x	g	g	x	x	x	x	x	x	x	x	x	x
Function name	x	x	x	x	x	x	x	x	x	x	x	g	g	x	x	x	x	x	x	x	x	x	x
Function parameter list	x	x	x	x	x	x	x	x	x	x	x	g	g	x	x	x	x	x	x	x	x	x	x
Function return value	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x	x
I/O error routine	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x	x
I/O object	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x	x
I/O option - ADD	c	c	x	c	c	c	c	c	c	c	x			c	c	c	c	c	c	c	c	c	x
I/O option - CLOSE	x	x	c	x	c	c	c	c	c	c	x			x	x	x	c	x	c	x	c	c	x
I/O option - CONVERSE	c		x	c	c	c		x		c	c			x	x	x	x	x	x	x	x	c	x
I/O option - DELETE	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	c	x
I/O option - DISPLAY	x	x	x	x	x	c	x	x	x	x	x			x	x	x	x	x	x	x	x		x
I/O option - EXECUTE	x	x	x	x	x	x	x	x	x	x	x	g	g	x	x	x	x	x	x	x	x	x	x

Table 8. Function elements (continued)

Element	COBOL											GUI		C++								Java	
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	Windows NT	Test Facility
I/O option - INQUIRY	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	c	x
I/O option - REPLACE	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	c	x
I/O option - SCAN	x	x	c	x	c	c	c	c	c	c	x			x	x	x	x	x	x	x	x	c	x
I/O option - SCANBACK	c	c	x	c	c		c	x	c	x	c			x	x	x	x	x	x	x	x		x
I/O option - SETINQ	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	c	x
I/O option - SETUPD	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	c	x
I/O option - SQLEXEC	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	c	x
I/O option - UPDATE	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	c	x
SQL statement	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x		x
SQL statement - Declare cursor with hold	c	c	x	x	x	x	x	c	c	x	x			x	x	x	x	x	x	x	x		x
SQL statement - Execution time statement build	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x		x

Table 8. Function elements (continued)

Element	COBOL											GUI		C++								Java		Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	Windows NT	
SQL statement - Model SQL statement generation	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x			x
SQL statement - Single row select	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x			x
SQL statement - UPDATE or SETUPD function name	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x			x

Note: *Includes Windows 95 and Windows 98, Windows NT, and Windows 2000.

Note: In this table, the following characters are used to indicate the level of support:

x Supported
c Supported with compatibility considerations
g The part or some of its features can be connected in a GUI application
blank Not supported

DL/I call

A DL/I call is created when a function uses an I/O object that is a segment in a DL/I database.

Uses

A default DL/I call is generated automatically based on the I/O option, the definition of the DL/I segment, and the definition of the database structure in the PSB part. The default call specification can be modified by the developer.

Target environments for DL/I call

See the following pages for the individual elements that make up the DL/I call definition.

DL/I call - Database identifier

Database identifier identifies the database in the program PSB that is to be accessed by this DL/I call.

Uses

Database identifier is a combination of program control block (PCB) number and database name. The number identifies which PCB in the program specification block (PSB) is to be used when the database name appears in more than one PCB in the PSB definition.

The default value is the database name for the first PCB in the PSB that contains a segment with the same name as the I/O object.

Target environments for Database identifier

Environment	Compatibility Considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
CICS for AIX	Not supported.

DL/I call - Database identifier

Environment	Compatibility Considerations
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
Test Facility	None.

DL/I call - Scan for update

Scan for update specifies whether a segment retrieved by a SCAN I/O option can be replaced or deleted.

Uses

If you do not specify scan for update, you cannot replace or delete the DL/I segment after a SCAN I/O option.

Target environments for Scan for update

Environment	Compatibility Considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	Not supported.

Environment	Compatibility Considerations
AIX	Not supported.
HP-UX	Not supported.
CICS for AIX	Not supported.
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
Test Facility	None.

DL/I call - Scan in parent

Scan in parent specifies whether the scan range of a DL/I call is limited to the parent chain of the database hierarchy.

Uses

If you do not specify Scan in parent, the next segment of that type in the database is retrieved regardless of the parent chain.

Target environments for Scan in parent

Environment	Compatibility Considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	Not supported.
OS/400	Not supported.

DL/I call - Scan in parent

Environment	Compatibility Considerations
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
CICS for AIX	Not supported.
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
Test Facility	None.

DL/I call - Segment search arguments

Segment search arguments (SSAs) identify the segments in the database to be accessed on a DL/I call. An SSA can also contain command codes that control the type of processing performed and qualification statements that specify search criteria for segment selection.

Uses

SSAs are automatically generated for you based on the I/O option and the position of the object segment in the database hierarchy in the PSB definition. You can modify the default SSAs to change the processing performed by the DL/I call.

You can enter the following information when modifying the SSA list:

Segment name
The name of the segment accessed by the SSA. The segment must be defined in the parent chain that goes from the object segment back to the root segment in the database hierarchy.

Command codes
Command codes are optional codes that identify special processing to be performed. Up to 4 codes can be entered in the command code column. Refer to the IMS documentation for a more detailed description of the command codes.

The following command codes are valid for SSAs:

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.

Segment field

A qualification statement consists of a segment field, a relational operator, and a comparison value item. The segment field identifies

the name of the field used for segment selection. You must specify the field name as defined in the DL/I database description.

When the program runs, DL/I compares the value in the segment field with the value in the comparison value item to determine whether the segment qualifies for selection.

The default value used in the generated SSA list is the name of either the segment's key item or the index key defined for the segment in the PSB. If both are defined, the name of the index key is the default. If neither key is defined, no qualification statement is generated for the segment.

Relational operator

The following relational operators are used for comparing the segment values and the Comparison Value Item:

EQ (=) Equal

NE (¬=)
Not equal

GT (>)
Greater than

GE (>=)
Greater than or equal

LT (<) Less than

LE (<=)
Less than or equal

Comparison Value Item

The Comparison Value Item is the name of an item in a record, table, or map. When the program runs, the value in this item is used as the field value in building the SSA for the DL/I call. The field 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 item name can be qualified and/or subscripted. Literals cannot be used for the item name. If no qualifier is specified, the segment name from the current SSA is used as the qualifier. If that segment does not contain the item, the I/O object name is used as the default qualifier.

The default value is the name of either the segment's key item or the index key defined for the segment in the PSB. If both are defined, the name of the index key is the default.

Both the Segment Field and the Comparison Value Item must have the same length. If the Segment Field is defined to the VisualAge Generator Developer, the preprocessor verifies that the lengths are equal. If the Segment Field is not defined to the generator, you are responsible for ensuring that the fields have equal length.

Boolean operator

The Boolean operator identifies the presence of an additional qualification statement and shows how the true or false values of the qualification statements are to be combined.

A Boolean operator in the continuation line indicates that there are additional qualifications for the SSA. On the continuation line, you leave the segment name field and command code field blank and enter data in the qualification statement fields only.

Valid Boolean operators are:

& or AND
AND operator

| or OR
OR operator

Note: Boolean operator "#:" (independent AND) is not supported. The "**" and "+" forms of the AND and OR operators are not supported.

For a segment to satisfy an SSA with multiple qualification statements, a segment can satisfy any set of qualification statements. A set consists of a sequence of qualification statements that are joined by AND operators. To satisfy the set, a segment must satisfy each of the qualification statements in the set. Each OR starts a new set of qualification statements.

No Boolean operators are used in the default SSA list built by the VisualAge Generator Developer.

Definition considerations for Segment search arguments

You can specify most parameters for a DL/I call. The only DL/I call parameters you cannot specify are the address of the I/O area and the function code. The generated program takes care of I/O area allocation for you. The function code is determined from the I/O option and the SCAN parameters.

The function codes used for each I/O option are the following:

I/O OPTION	SCAN FOR UPDATE	SCAN IN PARENT	DL/I FUNCTION
INQUIRY			Get Unique
UPDATE			Get Hold Unique
ADD			Insert
REPLACE			Replace

DELETE			Delete
SCAN	No	No	Get Next
SCAN	Yes	No	Get Hold Next
SCAN	No	Yes	Get Next in Parent
SCAN	Yes	Yes	Get Hold Next in Parent

Target environments for Segment search arguments

Environment	Compatibility Considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	Command codes M, R, S, W, and Z are not supported.
MVS batch	Same as MVS/TSO.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	Command codes C, M, P, R, S, W, and Z are not supported.
VSE batch	Same as CICS for VSE/ESA.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
CICS for AIX	Not supported.
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.

Environment	Compatibility Considerations
Test Facility	None.

Function

A function is built around a specific action called an I/O option. An I/O option is the I/O to be performed by a function, such as displaying a map or gaining access to a record.

You name the map or record used as the object of the function, which is called the I/O object, in the function specifications. You use only one I/O object per I/O option.

Uses

You can place additional statements in the function definition, either before or after the I/O option.

Target environments for function

Supported in all environments without compatibility considerations.

Function description

Function description is a text string from 1 to 30 characters that describes a function.

Uses

Function description is optional and does not affect execution.

Target environments for function description

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.

Environment	Compatibility Considerations
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Function local storage list

The local storage list shows the names and types of storage areas may that may be accessed solely by this function.

Uses

The following can be specified for each local storage area:

Name The name of the data item or record used as a local storage area.

Type The part type of the local storage area:

- Item
- Record

Description

The description of the local storage area. This is not the description of the shared data item in the library.

Item Usage

Item usage indicates whether the data item definition is stored as a separate data item or stored as part of the function definition.

Item usage can be set to the following:

- Nonshared - These characteristics apply only to the definition of the item in this function and the characteristics are stored with the function containing the item in its local storage list.

- Shared - These characteristics apply wherever a shared item with the same name is defined in any data structure. Shared characteristics are stored in a data item part, independent of the data structures, function local storage lists, or function parameter lists to which they belong.

Item Type

Data item type specifies the internal format or type of data. The data type determines how the item is processed when referenced in processing statements.

The following types of data are available:

Bin Binary number
CHA Character data
DBCS Double-byte character data
Hex Hexadecimal data
Mixed DBCS data mixed 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

Item Bytes

Item bytes specify the number of bytes required to store the data item internally.

Item Decimal Places

Item decimal places specifies the number of places reserved to the right of an implied decimal point. The default is 0 (no decimal places).

Definition considerations for function local storage list

For records, the local storage name must be the name of a working storage record part in the library. The part definition defines the storage layout of the local storage data.

Defining a local storage data item does not create a data item part in the library unless it is flagged as a shared item definition.

Local storage is not initialized upon entry into a function. Therefore, the user should make no assumptions as to any of the local storage data area values.

The scope of reference for a record or item named as local storage for a function is limited to that function only. If it is to be known by any other function, it must be passed to that function as an argument. If it is to be known to the caller, then it should not be a local storage definition. It should be a shared definition or be received via a parameter instead. The same record or item can be named in the local storage list for more than one function. Each function gets a separate copy of the storage mapped by the definition.

Definition of a local storage area that has the same name as a global program variable hides the global program variable from direct reference by the function. The function cannot modify the global variable in this case.

Target environments for function local storage list
Supported in all environments without compatibility considerations.

Function name

The function name identifies a set of logic that can perform an I/O operation.

Uses
See Appendix B. Naming conventions for data item, record, function names for function naming conventions.

Target environments for function name
Supported in all environments without compatibility considerations.

I/O error routine

I/O error routine is the name of an error handling subroutine. An error routine is started when an error occurs during execution of an I/O option that accesses a record.

Uses
If you do not specify an error routine, a program ends when an error occurs with a message describing the error condition. This includes standard situations such as the end-of-file (EOF) condition.

You cannot specify error routines for functions with map I/O objects or for EXECUTE functions. Display or printer errors cause the program to end.

The error routine can be any of the following:

- A valid special function word (EZERTN, EZEFLO, EZECLOS)
- The name of a function.

If the error routine is a main function, then control is transferred to that function when an error occurs and does not return to the failing I/O option. Otherwise, control returns to the statement following the I/O option after the error routine ends. When a function invoked as an error routine is defined to have a return value, the return value is ignored.

You can test error codes returned by the system using the TEST, WHILE, and IF statements.

Target environments for function error routine

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Function parameter list

The parameter list shows the names and types of parameters that are received by the function. The list is an ordered list of data areas to be accessed solely by this function.

Uses

The following can be specified for each parameter:

Name The name of the data item or record used as a parameter.

Type The part type of the parameter:
- Item
- Map Item
- SQL Item
- Record

Description
The description of the parameter. This is not the description of the shared data item in the library.

Item Usage
Item usage indicates whether the data item definition is stored as a separate data item or stored as part of the function definition.

Item usage can be set to the following:
- Nonshared - These characteristics apply only to the definition of the item in this function and the characteristics are stored with the function containing the item in its parameter list.
- Shared - These characteristics apply wherever a shared item with the same name is defined in any data structure. Shared characteristics are stored in a data item part, independent of the data structures, function local storage lists, or function parameter lists to which they belong.

Item Type
Data item type specifies the internal format or type of data. The data type determines how the item is processed when referenced in processing statements.

The following types of data are available:

Bin Binary number
CHA Character data
DBCS Double-byte character data
Hex Hexadecimal data
Mixed DBCS data mixed 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
ANYCHA
Character data of any length
ANYDBCS
Double-byte character data of any length
ANYHEX
Hexadecimal data of any length

ANYMIX
DBCS data mixed with single-byte character data of any length

ANYNUMERIC
Bin, Num, Numc, Pacf, or Pack data of any length with any number of decimal places

ANYUNICODE
Unicode data of any length

Item Bytes
Item bytes specify the number of bytes required to store the data item internally.

Item Decimal Places
Item decimal places specifies the number of places reserved to the right of an implied decimal point. The default is 0 (no decimal places).

Definition considerations for function parameter list

For records, the parameter name must be the name of a working storage record part in the library. The part definition defines the storage layout of the parameter received by the function.

Defining a parameter data item does not create a data item part in the library unless it is flagged as a shared item definition.

For items with one of the ANY item types; bytes and decimals are not allowed. For map item parameters, the only valid parameter item types are: NUM, CHA, DBCS, MIX, ANYNUMERIC, ANYCHA, ANYDBCS, and ANYMIX. For SQL item parameters, the only valid parameter item types are: BIN, CHA, DBCS, HEX, PACK, ANYNUMERIC, ANYCHA, ANYDBCS, and ANYHEX.

Defining item parameters as one of the ANY item types specifies loose typing of the parameter. The 4GL statements in the function operate on the parameter using the data item definition specified for the argument. ITF and the C++ generator implement loosely typed parameters by passing the arguments as item objects which contain the item definition as well as the item value. The COBOL generator implements loosely typed parameters by generating multiple copies of the function code, one copy for each unique combination of loosely typed argument definitions used within the program.

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 data type are either specified or defaulted for you. Test and generation will require exact matches between arguments and parameters when strong typing is used.

When map or SQL items are passed as arguments, the item state is available to the logic of the receiving function. This is so the user can test for and modify the state of the item. For example, TEST SQL-item TRUNC true,false; or SET map-item MODIFIED;. This may imply that the state of the map containing the map-item is updated as well. In order to test or set map conditions, the Parameter type must be Map Item. Likewise, in order to test or set SQL item conditions, the Parameter type must be SQL item. A parameter whose type is Map Item must be passed a map item as an argument. A parameter whose type is SQL Item must be passed an SQL item as an argument. Map or SQL items may be received into a parameter whose Parameter type is Item, but the specific map item state information and SQL item state information will not be available. An attempt to reference the state information in this case will result in an error in ITF and in the preprocessor.

EZEwords cannot be specified as parameters.

When records are passed as arguments, their level-77 items are not passed. Only the data structure is passed. The function receives a string of data and then accesses it using the parameter record definition. The length of the parameter record definition must be less than or equal to the argument record length. If the argument length is greater than the parameter length, the invoked function only has access to the amount of data defined by the parameter definition.

Functions may receive arrays as parameters only as part of a record. When a function that has no parameters defined is named as a map edit routine, the map array is available to the function by its map array name.

The scope of reference for a record or item named as a parameter for a function is limited to that function only. The same record or item can be named in the parameter list for more than one function. Each function gets a separate copy of the storage mapped by the definition.

Parameters are passed by reference. Therefore when a global variable is passed as an argument to a function and the function modifies the value of the parameter it received, then the value of the global variable has been modified. Definition of a parameter, local storage, or return value that has the same name as a global program variable hides the global program variable from direct reference by the function. The function cannot modify the global variable in this case.

Target environments for function parameter list

Supported in all environments without compatibility considerations.

Function return value

The return value defines the characteristics of a data area that is returned to the invoking function upon termination of this function. Any value specified on an EZERTN statement must be compatible with the characteristics defined. Each function has one and only one return value.

Uses

The following can be specified for the function return value:

Description
The description of the return value.

Type Type specifies the internal format or type of data. The data type determines how the return value is processed.

The following types of data are available:

Bin Binary number
CHA Character data
DBCS Double-byte character data
Hex Hexadecimal data
Mixed DBCS data mixed 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

Bytes Bytes specify the number of bytes required to store the data item internally.

Decimal Places
Decimal places specifies the number of places reserved to the right of an implied decimal point. The default is 0 (no decimal places).

Definition considerations for function return value

Return values are defined with strong typing. Bytes, decimals, and a specific data 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.

If a return value definition is specified, all EZERTN statements in that function must have an argument specified. If a return value definition is not specified, there may be EZERTN statements, but none of them may have an argument specified. If the logic of the function is such that the routine falls through to the end without executing an EZERTN statement, an implicit EZERTN is executed, returning a temporary storage area that is initialized to a default value according to the definition.

Target environments for function return value

Supported in all environments without compatibility considerations.

I/O object

I/O object is the name of a record or map accessed by the I/O option.

Uses

If the I/O option is EXECUTE, an I/O object is not allowed. If the I/O option is SQLEXEC, the I/O object is optional. All other I/O options require an I/O object.

Target environments for I/O object

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

I/O option

I/O option is the I/O operation in a function.

Uses

Only one option can be specified per function. EXECUTE is the default option.

The following are the I/O options that can be specified:

- ADD
- CLOSE
- CONVERSE
- DELETE
- DISPLAY
- EXECUTE
- INQUIRY
- REPLACE
- SCAN
- SCANBACK
- SETINQ
- SETUPD
- SQLEXEC
- UPDATE

Each option is described individually on the pages that follow.

Target environments for I/O option

The behavior of an I/O option varies with the type of file or database being accessed. The file type is determined from the record organization specified for the I/O object and the system file type associated with the record file during generation. Refer to the *VisualAge Generator Generation Guide* for more information on records and resource association files.

See the individual descriptions of the I/O options for variations in I/O option behavior based on target environment and file type.

I/O option - ADD

The ADD I/O option places a new record in a file, database, or message queue. The program should initialize all fields in the record before processing the ADD I/O option.

Uses

ADD is valid for DL/I segment, indexed, message queue, relative, serial and SQL row records.

Definition considerations for ADD

When you use an ADD function with a serial file, records are automatically appended to the end of the file. Exceptions are noted under the target platform compatibility considerations.

When you use an ADD function with a DL/I segment, the program should initialize the key fields of all segments that are parents of the segment being added.

When you use an ADD I/O option with an SQL row record, items in the record marked as read-only are not written to the database.

When you use the ADD I/O option to add a message queue record to a message queue, VisualAge Generator automatically specifies the MQSeries calls appropriate for the state of the queue:

MQCONN
: Connect the VisualAge Generator program to the default queue manager if no connection is active

MQOPEN
: Establish access to the queue if the queue is not open

MQPUT
: Put the message queue record in the queue

Some file types do not allow an ADD and SCAN function for the same serial file to be done in the same program. When using both an ADD and a SCAN function for a serial file in the same program, the file is closed and reopened whenever the program changes from adding to scanning or from scanning to adding. When the file is closed, file position is lost. Therefore, the first SCAN function after an ADD function reads the first record from the file. The following list identifies the file types that support both the ADD and SCAN I/O option in the same program:

File type	ADD and SCAN supported
GSAM	No
MMSGQ	No
OS2COBOL	Yes
SEQ	Yes
SEQRS	Yes
SMSGQ	No
SPOOL	No
TEMPAUX	Yes
TEMPMAIN	Yes

TRANSIENT Yes

VSAM Yes

VSAMRS Yes

Target environments for ADD

Environment	Compatibility Considerations
VM CMS	The first ADD function to a serial non-VSAM file adds data to the beginning of the file and all previous data is lost, unless the file is allocated using the DISP MOD option on the CMS FILEDEF command. Until the file is closed, subsequent ADD functions place data following the previously added data.
VM batch	Same as VM CMS.
CICS for MVS/ESA	None.
MVS/TSO	The first ADD function to a serial non-VSAM file adds data to the beginning of the file and all previous data is lost, unless the file is allocated using the MOD option on a TSO ALLOCATE command. Until the file is closed, subsequent ADD functions place data following the previously added data.
MVS batch	The first ADD function to a serial non-VSAM file adds data to the beginning of the file and all previous data is lost, unless the file is allocated using DISP=MOD in the JCL for the batch job. Until the file is closed, subsequent ADD functions place data following the previously added data. An ADD function for a serial record assigned to a GSAM file results in an ISRT command to the GSAM database. The program starts adding the records at the beginning of the file unless the file is allocated using DISP=MOD in the JCL for the batch job. If a variable-length serial record is in a file associated with GSAM and the record length is longer than the physical file, DL/I returns a blank status code. Data is truncated, but no message is issued because the situation cannot be detected.

Environment	Compatibility Considerations
IMS/VS	A serial record must be associated with an alternate PCB (a TP PCB in the PSB). The IMS message header (length, ZZ field, and transaction code) is automatically added to each record written to the message queue. An ADD function for a serial record assigned to a message queue results in an ISRT call to the message queue. If an error occurs and the record is assigned to a multiple segment message queue and associated with PCB #2 (the express PCB), any records already added are committed, even if an explicit CLOSE function has not occurred. If it is important that these records are not committed, include an additional express PCB in the PSB and associate the file with the additional express PCB.
IMS BMP	An ADD function to a serial non-VSAM file adds data to the beginning of the file and loses all previous data, unless the file is allocated using DISP=MOD in the JCL for the batch job. Until the file is closed, subsequent ADD functions place data following the previously added data. An ADD function for a serial record assigned to a message queue results in an ISRT call to the message queue. The IMS message header (length, ZZ field, and transaction code) is automatically added to each record written to the message queue. An ADD function for a serial record assigned to a GSAM file results in an ISRT to the GSAM database. The program starts adding the records at the beginning of the file unless the file is allocated using DISP=MOD in the JCL for the batch job. If a variable-length serial record is in a file associated with GSAM and the record length is longer than the physical file, DL/I returns a blank status code. Data is truncated, but no message is issued because the situation cannot be detected.
CICS for VSE/ESA	The first ADD function to a SPOOL file creates a new VSE/POWER queue part and adds the data to the beginning of the file. Until the file is closed, subsequent ADD functions place data following the previously added data. Once a SPOOL file that is a VSE/POWER LST or PUN queue part is closed, a subsequent ADD function to that file creates a new segment for that queue part. Once a SPOOL file that is a VSE/POWER RDR queue part is closed, a subsequent ADD function to that file creates a new RDR queue part that is processed as a separate batch job.
VSE batch	Same as CICS for VSE/ESA.
CICS for OS/2	Use with message queue records is not supported.
OS/400	None.

Environment	Compatibility Considerations
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	If the file is a native serial file and the /REPLACE option was specified for the file in the resource association file, the first ADD adds the record to the beginning of the file. All previous data in the file is lost.
AIX	Same as OS/2 (C++).
HP-UX	Same as OS/2 (C++).
CICS for AIX	Same as OS/2 (C++).
Windows NT (C++)	Same as OS/2 (C++).
Windows NT (Java)	Serial, SQL, and message records are the only supported I/O objects.
CICS for Windows NT	Same as OS/2 (C++).
Solaris	Same as OS/2 (C++).
CICS for Solaris	Same as OS/2 (C++).
Test Facility	None.

I/O option - CLOSE

The **CLOSE** I/O option can do any of the following:

- Close a file
- Disconnect a printer
- Release any unprocessed rows in a set of SQL row records selected by the UPDATE, SETUPD, or SETINQ I/O option
- Close a message queue

Uses

CLOSE is valid for indexed, message queue, relative, serial or SQL row records and for printer maps.

Definition considerations for CLOSE

The function of the CLOSE I/O option varies with the type of I/O object and the system on which a CLOSE is issued.

Files If the I/O object is a file, the CLOSE I/O option results in system close for the file. If you use EZEDEST to change the resource name associated with a file that is currently open, that file is closed on the next I/O option issued for the file. You do not need to explicitly specify the CLOSE I/O option for the previously opened file.

If the program ends before all files are closed, VisualAge Generator Server for MVS, VSE, and VM and VisualAge Generator Server ensure that all files are closed.

Message queue

If the I/O object is a MQSeries message queue, the CLOSE I/O option closes the queue.

Printer maps

If the I/O object is a printer map, the CLOSE I/O option issues a form feed and then either disconnects from the printer or closes the printer file on systems where the print lines are spooled to a file.

When you use EZEDESTP to change the print destination, use a CLOSE I/O option to close the print file specified by the current value of EZEDESTP. Issue a CLOSE I/O option for each destination you use, because multiple print files can be open at the same time.

If the program ends before all printers are closed, VisualAge Generator Server for MVS, VSE, and VM and VisualAge Generator Server ensure that all printers are closed.

SQL row record

If the object is an SQL row record, a CLOSE I/O option results in a CLOSE cursor when an SQL cursor is open for the record.

Cursors that were declared using CURSOR WITH HOLD are not closed on a COMMIT. But a rollback or a CONNECT function will close all cursors including those declared using a WITH HOLD.

The CLOSE function for an SQL row record is performed automatically when:

- The SCAN loop following a SETINQ or SETUPD function continues until a no record found (NRF) condition is encountered, indicating all rows in the set were processed.
- A single row is read for an INQUIRY function.
- A REPLACE or DELETE function for the same I/O object is executed following an UPDATE function.
- Another INQUIRY, UPDATE, SETINQ, or SETUPD function is executed for the same I/O object. Only one set of rows can be selected for a specific SQL row record at a time.
- A program transfers to another program.
- Database changes are committed or rolled back.

Target environments for CLOSE

Environment	Compatibility Considerations
VM CMS	None.

Environment	Compatibility Considerations
VM batch	None.
CICS for MVS/ESA	For SPOOL files, the SPOOL CLOSE command is executed for the file. For all other files, the CLOSE function does not physically close a file, it resets the position pointer to the beginning of the file. The CLOSE I/O option does not delete temporary storage files. Automatic CLOSE processing is performed when a segmentation break occurs at a CONVERSE as well as at the end of a program.
MVS/TSO	None.
MVS batch	A CLOSE function for a record or printer map assigned to a VSAM or MVS sequential file results in an OS CLOSE for the file. A CLOSE function for a serial record assigned to a GSAM file results in a CLSE call to the GSAM database. A CLOSE function for a printer map assigned to a GSAM file results in a form feed followed by a CLSE call.
IMS/VS	A CLOSE function for a serial record assigned to a message queue for output results in a PURG call to the message queue. A CLOSE function for a serial record assigned to a message queue for input is ignored. A CLOSE function for a printer map results in a form feed followed by a PURG call. When a main program ends or when a program called by a non-VisualAge Generator program ends, a form feed is issued for each destination to which printer maps were sent, followed by a PURG call for that destination. A main program is considered to have ended when it finishes its last function or does an EZECLOS, an XFER, or a DXFR. A form feed and a PURG call are also issued for each destination when a segmentation break occurs at a CONVERSE function. A form feed and CLOSE function are not done when a program called by another VisualAge Generator program ends.

Environment	Compatibility Considerations
IMS BMP	A CLOSE function for a record or printer map assigned to a VSAM or MVS sequential file results in an OS CLOSE for the file. If the GSAM file is open due to an I/O function other than CLOSE, a CLOSE function for a serial record assigned to a GSAM file results in a CLSE call to the GSAM database. A CLOSE function for a printer map assigned to a GSAM file results in a form feed followed by a CLSE call. A CLOSE function for a serial record assigned to a message queue for output results in a PURG call to the message queue. A CLOSE function for a serial record assigned to a message queue for input is ignored. A CLOSE function for a printer map assigned to a message queue results in a form feed followed by a PURG call. When a main program ends or when a program called by a non-VisualAge Generator program ends, a form feed is issued for each message queue destination to which printer maps were sent, followed by a PURG call for that destination.
CICS for VSE/ESA	For SPOOL files, a CLOSE function results in a CLOSE VSE/POWER access service request for that part. For all other files, a CLOSE function does not physically close a file, it resets the position pointer to the beginning of the file. A CLOSE function does not delete temporary storage files. Automatic CLOSE processing is performed when a segmentation break occurs at a CONVERSE function as well as at the end of a program.
VSE batch	For SPOOL files, a CLOSE function results in a CLOSE VSE/POWER access service request for that part. A CLOSE function for a record or printer map assigned to a file with the type SEQ, VSAM, or VSAMRS results in a system close for the file.
CICS for OS/2	For all files, the CLOSE function does not physically close a file, it resets the position pointer to the beginning of the file. The CLOSE function option does not delete temporary storage files. Automatic CLOSE processing is performed when a segmentation break occurs at a CONVERSE as well as at the end of a program. Using CLOSE with a message queue record is not supported.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.

Environment	Compatibility Considerations
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	CLOSE for CICS files does not physically close the file, it resets the position pointer to the beginning of the file. The CLOSE does not delete temporary storage files.
Windows NT (C++)	None.
Windows NT (Java)	Serial, SQL, and message records are the only supported I/O objects.
CICS for Windows NT	Same as CICS for AIX.
Solaris	None.
CICS for Solaris	CLOSE for CICS files does not physically close the file, it resets the position pointer to the beginning of the file. The CLOSE does not delete temporary storage files.
Test Facility	None.

I/O option - CONVERSE

If the program type is Web transaction, CONVERSE sends a user interface (UI) record and waits for input from the program user. If the program type is not Web transaction, CONVERSE displays a map and waits for input from the program user.

CONVERSE is valid only for display maps (3270 user interface screens) or UI records.

Using the CONVERSE I/O option with a map

If the CONVERSE I/O option sends a map, edit validation is bypassed if the program user presses an attention key or a function key defined as a bypass edit key.

Using the CONVERSE I/O option with a UI record

If the CONVERSE I/O option sends a UI record, all edits are bypassed if the submit value sent back is defined as a Submit Bypass item.

Definition considerations for CONVERSE with maps

If the CONVERSE I/O option sends a map and the program user presses Enter or a function key, the data entered by the program user is read and validated as specified in the map variable field edit definitions. If the data

entered is not valid, the map appears again without passing the input to the program for processing. A message prompts the user to correct the data in error.

Definition considerations for CONVERSE with UI records

If the CONVERSE I/O option sends a UI record and the program user submits the HTML page to the server:

- The specified edit validations occur at the Web Server where the UI Record beans have been deployed:
  - If the specified edit validations fail, the HTML page is sent back to the user. The HTML page can access error messages available in the UI bean. The default generated HTML page will show the error directly underneath the field in error.
  - If the specified edit validations succeed, the data is passed back to the program and any user defined edit functions are run on the server. If any of the user defined edit functions fail, the CONVERSE of the UI record is repeated, otherwise the program continues on after the CONVERSE statement.

Target environments for CONVERSE

Environment	Compatibility Considerations
VM CMS	UI records are not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	UI records are not supported.
MVS batch	UI records are not supported.
IMS/VS	Multiple partial maps cannot be used for terminals because the screen is erased before the map displays. **PA2** cannot be used as a bypass edit key.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	UI records are not supported.
OS/400	UI records are not supported.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.

Environment	Compatibility Considerations
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	UI records are the only supported I/O objects.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

I/O option - DELETE

DELETE removes a record from a file or database.

Uses

DELETE is valid for relative, indexed, DL/I segment and SQL row records.

You must first obtain the record by an UPDATE function or a SCAN for an update function for DL/I or relational databases.

Target environments for DELETE

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.

Environment	Compatibility Considerations
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	SQL records are the only supported I/O objects.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

I/O option - DISPLAY

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

Uses

If a map is sent to a terminal output buffer, the buffer contents are sent to the screen when the next CONVERSE occurs.

The DISPLAY option serves the following two purposes:

- Sends a map to a printer
- Sends a number of maps to the screen at once. The maps can be floating maps or fixed maps, each of which only partially fills the screen. Each DISPLAY option sends a map to the terminal I/O buffer until the CONVERSE option of a subsequent map causes all the accumulated maps (including the conversed map) to be sent to the screen.

DISPLAY is valid for both terminal and printer maps.

Target environments for DISPLAY

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.

Environment	Compatibility Considerations
MVS batch	None.
IMS/VS	The DISPLAY I/O option is only supported for printer maps.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	Not supported.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

I/O option - EXECUTE

EXECUTE is not associated with an I/O operation. EXECUTE has no I/O object (map or record).

Uses

Use EXECUTE for special processing, such as controlling the flow between functions, initialization, processing not to be repeated in an I/O function, error handling, and processing that ends the program.

Target environments for EXECUTE

Supported in all environments without compatibility considerations.

I/O option - INQUIRY

INQUIRY reads a single record from a file or database. The current value in the key identifies the record to be read.

Uses
INQUIRY is valid for indexed, relative, DL/I segment, or SQL row records.

Definition considerations for INQUIRY
For an SQL row record, the SELECT statement built for the INQUIRY function is always issued using an SQL cursor unless Single row select is specified for the SQL statement. The INQUIRY function reads the first row returned by the SELECT and automatically issues a CLOSE function to release any other rows.

Target environments for INQUIRY

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	SQL records are the only supported I/O object.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.

Environment	Compatibility Considerations
Test Facility	None.

I/O option - REPLACE

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

Uses

REPLACE is valid for indexed, relative, DL/I segment, or SQL row records.

You must first obtain the record by an UPDATE or SCAN for update function for DL/I or relational databases.

Definition considerations for REPLACE

The default SQL statement built for a REPLACE function for an SQL row record does not write to the database any item specified as the default key or as read-only in the SQL row record.

Target environments for REPLACE

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.

Environment	Compatibility Considerations
Windows NT (C++)	None.
Windows NT (Java)	SQL records are the only supported I/O objects.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

I/O option - SCAN

The **SCAN** I/O option reads the next record in a file, database or message queue.

Uses

The SCAN I/O option is valid for DL/I segment, indexed, message queue, relative, serial or SQL row records.

Definition considerations for SCAN

Any successful file I/O sets the position for the SCAN function. The position following an unsuccessful I/O operation is undefined. The program must establish file position again when an unsuccessful read occurs.

A `SET record SCAN` statement also establishes positioning to the next record to be retrieved if followed by a SCAN I/O option. The `SET record SCAN` statement is used only with indexed or DL/I segment records. It is not allowed for relative records and is ignored for all other record types.

DL/I segments

The scan position for records in a DL/I database depends on previous calls to the database. The Get Next in Parent option for the DL/I CALL defined for the function controls when EOF is returned for the SCAN. If Get Next in Parent is specified, EOF is indicated after the last segment of that type for the current parent has been read; if Get Next in Parent is not specified, EOF is indicated after the last segment of that type in the database has been read. Refer to the section on developing DL/I programs in the *Design Guide* online document for a more detailed discussion of the scan position for DL/I segments.

Message queue records

If you use the SCAN I/O option to read a message queue record in a message queue, VisualAge Generator automatically:

1. Connects to the queue manager, if the queue manager is not already connected

2. Opens the queue, if the queue is not already open
3. Gets the next message from the queue and moves the message contents to the message queue record structure

Relative and indexed files

The SCAN function reads the record following the last read record in key sequence. The first record of a file is read if no function that uses the record as an object has been previously executed.

A SCAN function following a SCANBACK function retrieves the record following the record accessed on the SCANBACK function. If a SCANBACK function returns an EOF condition, the SCAN function returns the first record in the file.

An EOF condition is returned on the SCAN function after the last record is read. For compatibility with versions of Cross System Product, relative file I/O will also return NRF on a SCAN function past the end of the file.

When using alternate indexes, a SCAN function returns the record in the file with the next higher alternate key than the current position in the file. A DUP condition occurs if the record retrieved using the SCAN function has the same key as another record in the file. An exception occurs when retrieving the last record in a group of duplicate-keyed records. In this case, although the record has a duplicate key, the DUP mnemonic is not set. If records with duplicate keys exist in the file, a SCAN function following a SCAN retrieves any duplicate-keyed record before retrieving the record with the next key. Records with duplicate keys are returned in the order that VSAM returns them. A SCAN function following a successful I/O option (other than a SCAN function that retrieved a duplicate-keyed record) skips over any remaining duplicate-keyed records and retrieves the record with the next greater key.

An EOF condition is returned on the SCAN after the last record has been read. For compatibility with previous versions of VisualAge Generator, relative file I/O will also return NRF on a SCAN past the end of the file.

Serial files

A SCAN function reads the record following the last record read in the entry sequence. The first record is read for the first scan of a file.

If the record accessed on the previous I/O operation was the last record in the file, SCAN returns EOF. The following list identifies the file types that support both the ADD and SCAN I/O option in the same program:

File type	ADD and SCAN supported

GSAM	No
MMSGQ	No
OS2COBOL	Yes
SEQ	Yes
SEQRS	Yes
SMSGQ	No
SPOOL	No
TEMPAUX	Yes
TEMPMAIN	Yes
TRANSIENT	Yes
VSAM	Yes
VSAMRS	Yes

When using both an ADD and a SCAN function in the same program, the file is closed and reopened whenever the program changes from adding to scanning or from scanning to adding. When the file is closed, the file position is lost; therefore, the first SCAN after an ADD will read the first record from the file.

SQL row records

The SCAN function reads the next row from a set of rows selected from the relational database by a SETINQ or SETUPD I/O option. If a row was selected using SETUPD, it can be replaced (REPLACE option) or deleted (DELETE option) immediately following the SCAN function that retrieved the row.

The SET record SCAN statement has no effect on SQL row records. Position for SQL row can only be set with a SETINQ or SETUPD I/O option.

A NRF condition is set if the last row in the set was retrieved on a previous SCAN function.

The scan position is lost if a CLOSE function is performed for the set of rows. See the description of the CLOSE I/O option for a description of when CLOSE processing is performed.

For details concerning SQL options, refer to the section on developing SQL programs in the *Design Guide* document.

Target environments for SCAN

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	The scan position is lost when a commit or rollback is issued, or following a CONVERSE function if running in segmented mode. Cursors that were declared using CURSOR WITH HOLD are not closed on a commit, but a rollback or database connect function will close all cursors including those declared using WITH HOLD.
MVS/TSO	None.
MVS batch	A SCAN function for a serial record assigned to a GSAM file results in a get next call to the GSAM database. If a variable-length serial record is in a file associated with GSAM and the record length is longer than the physical file, DL/I returns a blank status code. Data is truncated, but no message is issued because the situation cannot be detected.
IMS/VS	A serial record must be associated with the I/O PCB (PCB 0). The SCAN function is not supported for a transaction program or for a batch program that is called from a transaction program. Batch programs can use only one serial file for input. The IMS message header (length, ZZ field, and transaction code) is automatically removed from each record read from the queue. A SCAN function for a serial record assigned to a single-segment message queue results in a get unique (GU) call to the I/O PCB. This GU call results in an automatic commit point. The first SCAN for a serial record assigned to a multiple-segment message queue results in a GU call to the I/O PCB. Subsequent SCAN functions result in get next calls until an NRF (status code QD) condition is reached. The first SCAN function after the NRF results in another GU call, and the function continues until an EOF (status code QC) is reached. Each GU call results in an automatic commit point. During any specific scheduling of a batch program, the program can do a SCAN function from only one message queue. The transaction code for which IMS scheduled the program determines the message queue that is scanned. The system resource specified during generation is ignored.

Environment	Compatibility Considerations
IMS BMP	For an IMS batch-oriented BMP, a SCAN function for a serial record assigned to a GSAM file results in a get next call to the GSAM database. If a variable-length serial record is in a file associated with GSAM and the record length is longer than the physical file, DL/I returns a blank status code. Data is truncated, but no message is issued because the situation cannot be detected. For an IMS transaction-oriented BMP, a SCAN function for a serial record assigned to a message queue is the same as IMS/VS.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	The SCAN function is not supported for a serial record that is assigned to a SPOOL file.
CICS for OS/2	Same as CICS for MVS/ESA except that use with message queue records is not supported.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	Serial, SQL, and message records are the only supported I/O objects.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Examples for SCAN

Consider a file where the keys are as follows:

```
1, 2a, 2b, 2c, 3, 4
```

Where a, b, and c are used to indicate duplicate-keyed records for key 2 and the order in which they were added to the file.

The following examples illustrate the order in which records are retrieved.

Example 1:

I/O option	Key	Retrieves	Sets
INQUIRY	2	2a	DUP
SCAN		3	

Example 2:

I/O option	Key	Retrieves	Sets
SET record SCAN	2		
SCAN		2a	DUP
SCAN		2b	DUP
SCAN		2c	
SCAN		3	

I/O option - SCANBACK

SCANBACK reads the previous record in an indexed file.

Uses

SCANBACK is valid only for indexed records.

Definition considerations for SCANBACK

A SCANBACK function returns the record in the file with the highest key that is less than the current position in the file. The last record of a file is read if no function that uses the record as an object has been previously executed and no `SET record SCAN` statement has been done.

A SCANBACK function on an uninitialized file causes an NRF condition to occur. A SCANBACK function on an empty file causes an EOF for a non-CICS environment, and both an EOF and an NRF for a CICS environment. An uninitialized file is one that has never had any records added to it. An empty file is one from which all records have been deleted.

The file position after an unsuccessful INQUIRY, UPDATE, SCAN, or SCANBACK function is undefined. The program must establish file position again when an unsuccessful read occurs.

The SCANBACK position is set on any successful I/O to the file. A SCANBACK function after any successful I/O operation retrieves the record with the highest key value that is less than the key of the record accessed on the previous I/O operation.

A SCANBACK function following a `SET record SCAN` statement retrieves the record with the highest key value that is less than or equal to the current record key value. A `SET record SCAN` with a key value set to all hexadecimal FF bytes prior to a SCANBACK function sets the position pointer in all environments to the end of the file so that the next SCANBACK function retrieves the last record in the file.

If a SCANBACK function follows a SCAN function that returned an EOF condition, the last record in the file is retrieved.

An EOF condition occurs if no previous record was in the file. This occurs, for example, when SCANBACK functions are repeated past the beginning of the file.

When using alternate indexes, a SCANBACK function returns the record in a file with the highest alternate key that is less than the current position in the file. A DUP condition occurs if the record retrieved using a SCANBACK function has the same key as another record in the file. An exception occurs when retrieving the last record in a group of duplicate-keyed records. In this case, although the record has a duplicate key, the DUP mnemonic is not set. If records with duplicate keys exist in the file, a SCANBACK function following a SCANBACK function retrieves any duplicate-keyed record before retrieving the record with the previous key. Records with duplicate keys are returned in the order that VSAM returns them.

A SCANBACK function following a successful I/O option (except for a SCANBACK function that retrieved a duplicate-keyed record) skips over any remaining duplicate-keyed records and retrieves the record with the next lower key.

Target environments for SCANBACK

Environment	Compatibility Considerations
VM CMS	VSAM files that use a SCANBACK function must be specified as file type VSAMRS during generation. All programs within a run unit that share the file must also specify VSAMRS for the file.
VM batch	Same as VM CMS.
CICS for MVS/ESA	None.
MVS/TSO	Same as VM CMS.
MVS batch	Same as VM CMS.
IMS/VS	Not supported.
IMS BMP	Same as VM CMS.

Environment	Compatibility Considerations
CICS for VSE/ESA	None.
VSE batch	Same as VM CMS.
CICS for OS/2	None.
OS/400	SET Record SCAN must be used before SCANBACK if the SCANBACK is the first I/O operation performed on the file.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	Not supported.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Examples for SCANBACK

Consider a file where the keys are as follows:

```
1, 2a, 2b, 2c, 3, 4
```

Where a, b, and c are used to indicate duplicate-keyed records for key 2 and the order in which they were added to the file.

The following examples illustrate the order in which records are retrieved.

Example 1:

I/O option	Key	Retrieves	Sets
INQUIRY	3	3	
SCANBACK		2a	DUP
SCANBACK		2b	DUP

SCANBACK	2c
SCANBACK	1

Example 2:

I/O option	Key	Retrieves	Sets
SET record SCAN	2		
SCAN		2a	DUP
SCAN		2b	DUP
SCANBACK		1	

Example 3:

I/O option	Key	Retrieves	Sets
SET record SCAN	1		
SCANBACK		nothing	EOF

I/O option - SETINQ

SETINQ selects a set of rows from a relational database for later retrieval with the SCAN I/O option.

Uses

The object must be an SQL row record.

Definition considerations for SETINQ

The default SQL statement built for a SETINQ function selects all rows that meet any default selection conditions defined for the SQL row record, and whose key column is greater than or equal to the current key item value. The rows are sorted in key column sequence if a key was specified.

The default WHERE clause is not built when multiple-column keys exist.

Target environments for SETINQ

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.

Environment	Compatibility Considerations
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	SQL records are the only supported I/O object..
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

I/O option - SETUPD

SETUPD selects a set of records from a relational database for later processing with the SCAN I/O option.

Uses

The selected records can be replaced or deleted. The object must be an SQL row record.

Definition considerations for SETUPD

If a single key is defined for an SQL row record, the default SQL statement built for a SETUPD function selects all rows that meet any default selection conditions defined for the SQL row record, and whose key column is greater than or equal to the current key item value. The rows are not sorted.

The default WHERE clause is not built when multiple-column keys exist. If multiple keys are defined for the SQL row record, the default SQL statement retrieves all rows that meet the default selection conditions defined for the record.

Target environments for SETUPD

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	SQL records are the only supported I/O objects..
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

I/O option - SQLEXEC

SQLEXEC enables you to define your own SQL statement to run as the I/O option.

Uses

SQLEXEC is valid only for SQL records. However, using an SQL row record I/O object is optional.

Definition considerations for SQLEXEC

The SQLEXEC I/O option supports the use of SQL data manipulation and definition statements that are not directly supported by other I/O options.

The statements supported by SQLEXEC are as follows:

- Multirow INSERT
- Multirow DELETE
- Multirow UPDATE
- GRANT
- REVOKE
- CREATE
- DROP
- SET

To use the SQLEXEC function you must be familiar with SQL statement syntax. Refer to the SQL reference manual for the relational database manager used at your location for information on SQL statement syntax. The rules for SQL statement syntax differ among the various database managers.

SELECT statements cannot be issued using the SQLEXEC I/O option because the SQL interfaces that support the SQLEXEC I/O option do not support SELECT statements. Use the SETINQ, SETUPD, INQUIRY, or UPDATE I/O options for SELECT processing.

If you do not enter an SQL statement, the I/O behaves like an EXECUTE I/O option.

An I/O object is not required. You can specify an SQL row record as an I/O object.

Target environments for SQLEXEC

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.

Environment	Compatibility Considerations
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	SQL records are the only supported I/O objects.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

I/O Option - UPDATE

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

Uses

UPDATE locks the record, protecting it from updates by other users, until another operation is complete for the file or database.

UPDATE is valid for indexed, relative, DL/I segment, or SQL row records.

Definition considerations for UPDATE

If the file or database is shared by multiple users, an UPDATE function should not be held across a CONVERSE function. This can cause other users attempting to access the record to wait until the first user responds to the CONVERSE function.

The default SQL statement built for an UPDATE function for an SQL row record reads the row whose key columns are equal to the current key item values. All columns represented in the record are retrieved; only columns other than key columns or read-only columns can be written back to the database on the associated REPLACE I/O option.

Target environments for UPDATE

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	SQL records are the only supported I/O objects.
CICS for Windows NT	None.

Environment	Compatibility Considerations
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

SQL statement

An SQL statement is created for a function to access a relational database.

Uses

A default SQL statement is generated for the function based on the I/O option and the definition of the SQL row record I/O object.

Definition considerations for SQL statement

You can modify the SQL statements for some I/O options if you understand SQL syntax. You cannot change the following:

- The table name clause in the SQL statement for any I/O option other than the SQLEXEC I/O option
- The SQL statements for the DELETE, SCAN, and CLOSE I/O options
- The WHERE CURRENT OF CURSOR clause in the SQL statements for the REPLACE and DELETE I/O options.

You can define entire SQL statements for the SQLEXEC function or use the model option to create default UPDATE or DELETE statements. You can modify all clauses in the model statement.

If you change or enter a statement, use the SQL statement syntax described in the appropriate DB2 reference manual with the following additions or exceptions:

- To use data items as host variables in the statement, place a colon immediately preceding the data item name.
- Do not use null indicator variables. Null indicators are maintained by the VisualAge Generator Developer for all items in SQL row definitions. Use the TEST and IF statements to test null indicators and the SET statement to set null indicators for SQL row items.
- Use an INTO clause with all SELECT statements. The SELECT might actually be executed with an SQL cursor. If so, the INTO clause identifies the data items that receive the data when a row is retrieved with the FETCH command associated with the cursor. The INTO clause is defined with the SELECT because a one-to-one relationship must be maintained between the selected columns and the items in the INTO clause. You can avoid use of a cursor for an INQUIRY I/O option by selecting the Single row select option.

- To enter a comment line in the statement, type /* as the first characters in the comment.
- Enter an SQL column name directly, or enter the data item name in the SQL row record preceded by an exclamation mark (!item-name). When the SQL statement is prepared for execution, the data item name is replaced by the SQL column name defined for the data item in the SQL row definition.

The SQLEXEC I/O option is used for advanced SQL programming functions for database manipulation. With SQLEXEC, you define the entire SQL statement. You can enter any statement that you execute using the EXECUTE command of the SQL interface for high-level languages.

For more information and examples of how to use SQL statements in a program, refer to the *Design Guide* document.

If the program contains a large number of SQL I/O options, DB2 precompiler limits can be exceeded. If you exceed a limit, split the program.

Target environments for SQL statement

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.

Environment	Compatibility Considerations
Windows NT (C++)	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

SQL statement - Declare cursor with hold

Declare cursor with hold specifies that the WITH HOLD clause is added to the DECLARE CURSOR statement that is issued for the SETINQ or SETUPD I/O options.

Uses

If you use the WITH HOLD clause, the cursor is not closed when a commit occurs.

The WITH HOLD clause is not effective on rollback functions or at the end of a segment.

Definition considerations for Declare cursor with hold

To avoid an SQL error, do the following:

- When using the SETUPD I/O option, specify the SCAN I/O option after a commit before using the DELETE or REPLACE I/O option,
- Before connecting to a different database using EZECONCT, use the CLOSE I/O option to close all cursors.

If you have specified an I/O option other than SETINQ or SETUPD, you cannot specify Declare cursor with hold.

Target environments for Declare cursor with hold

Environment	Compatibility Considerations
VM CMS	Refer to the reference manual for your version of SQL/DS VM to determine if the WITH HOLD clause can be specified on the DECLARE CURSOR statement. If the WITH HOLD clause is not supported, do not specify Declare cursor with hold. Otherwise, the SQL/DS VM precompiler fails when you prepare the program.
VM batch	Same as VM CMS.
CICS for MVS/ESA	None.

Environment	Compatibility Considerations
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	Refer to the reference manual for your version of DB2 VSE (SQL/DS) to determine if the WITH HOLD clause can be specified on the DECLARE CURSOR statement. If it is not supported, do not specify Declare cursor with hold. Otherwise, the DB2 VSE(SQL/DS) precompiler will fail when you prepare the program.
VSE batch	Same as CICS for VSE/ESA.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

SQL statement - Execution time statement build

Execution time statement build indicates that the SQL statement generated for a function is prepared dynamically each time it is executed. This lets you modify parts of the SQL statement (the WHERE clause, for example) at execution time.

Uses

Execution time statement build can be specified for SELECT statements and statements issued using the SQLEXEC I/O option.

Use Execution time statement build to do the following tasks:

- Dynamically modify an SQL statement generated for an SQL function when the program runs
- To use host variables in SQL where host variables are not normally supported.

If you do not specify Execution time statement build, the statement is built as a static SQL statement or prepared and executed using the dynamic or extended dynamic PREPARE and EXECUTE interface. You use host variables as defined in normal SQL statement syntax. All valid host variable data types are supported.

If you specify Execution time statement build, the statement is prepared each time the function is executed. SQLEXEC functions are executed using the SQL dynamic EXECUTE IMMEDIATE command. INQUIRY, SETINQ, UPDATE, and SETUPD functions are executed using PREPARE and cursor manipulation statements. Any REPLACE and DELETE functions associated with dynamic SELECT statements are also executed dynamically.

Definition considerations for Execution time statement build

When you specify Execution time statement build, the statement executed is built by replacing all the host variables in the statement (except host variables in the INTO clause in the SELECT statement) with the character representation of the contents of the host variables.

Only host variables with type CHA, BIN, or PACK can be used in the statement, except in the INTO clause. The CHA fields are inserted directly into the statement without being enclosed in single quotes. This has the advantage of allowing host variables to be used in places where SQL does not normally support host variables.

For example, you could code a host variable in place of an entire WHERE clause and have the program dynamically build the WHERE clause in the host variable at execution time.

Target environments for Execution time statement build

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.

Environment	Compatibility Considerations
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

SQL statement - Model SQL statement generation

Model SQL statement generation specifies whether a model SQL statement is generated for an SQLEXEC function, and if required, the type of model SQL statement.

Uses

You can specify one of the following:

None To define a function with an SQLEXEC I/O option without a model SQL statement.

Update

To define a function with an SQLEXEC I/O option with a model SQL statement for updating a table row.

The model SQL statement is derived from the SQL statement you specified as the default for the SQL row record that is the I/O object.

Delete To define a function with an SQLEXEC I/O option with a model SQL statement for deleting an SQL table row.

The model SQL statement is derived from the SQL statement you specified as the default for the SQL row record that is the I/O object.

Target environments for Model SQL statement generation

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

SQL Statement - Single row select

Single row select specifies single row selection if you are defining a function with an INQUIRY I/O option.

Uses

Single row select is designated during SQL statement definition for an INQUIRY function to indicate that the row be retrieved with a Single row select rather than with an SQL cursor.

This option is effective only with static execution and is ignored when the program is run in the test facility, which runs in dynamic mode.

Definition considerations for Single row select

Single row select is more efficient than retrieving a single row with a cursor, but it will fail if more than one record meets the selection criteria. Use Single row select when retrieving rows by key where the key is unique.

Target environments for Single row select

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
CICS for Windows NT	None.
Solaris	None.

Environment	Compatibility Considerations
CICS for Solaris	None.
Test Facility	Not supported.

SQL statement - UPDATE or SETUPD function name

UPDATE or SETUPD function name identifies the name of an UPDATE or SETUPD function that selected the rows to be replaced by a REPLACE I/O option.

Uses

The UPDATE or SETUPD function name is required for generation if more than one UPDATE or SETUPD function exists with the same I/O object as the REPLACE function and at least one of the FOR UPDATE OF clauses was modified.

The UPDATE or SETUPD function name provides the information needed to correctly associate SQL SELECT and UPDATE statements in the generated module.

Target environments for UPDATE or SETUPD function name

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.

Environment	Compatibility Considerations
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Chapter 4. Records

A record defines the organization and item structure of the record along with other options such as file name and record ID item. Record specification options vary depending on the organization you choose.

VisualAge Generator supports the following record organizations:
- DL/I segment
- Indexed
- Message queue
- Redefined
- Relative
- Serial
- SQL row
- User interface
- Working storage

Records are included in a program by specifying the record name as:
- An I/O object
- An entry in the table and additional records list for the program
- A working storage record
- A called parameter
- A parameter for a function
- A local storage area for a function
- The name of a record part included in a GUI definition

Record elements

Table 9. Record elements

Element	COBOL											GUI		C++								
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	Test Facility
Alternate specification	c	c	c	c	c	x	c	c	c	c	x			x	x	x	c	x	c	x	c	x

Table 9. Record elements (continued)

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
Default key item (SQL)	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Default selection conditions (SQL)	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
File name	c	c	c	c	c	c	c	c	c	c	x			x	x	x	c	x	c	x	c	x
Key item (DL/I)			x	x	x	x	x	x	x	x												x
Number of occurrences item	c	c	c	c	c	c	c	c	c	c		g	g	x	x	x	c	x	c	x	c	x
Organization - DL/I segment			x	x	x	x	x	x	x													x
Organization - Indexed	x	x	x	x	x		x	x	x	x	x			x	x	x	x	x	x	x	x	x
Organization - Message queue	x	x	x	x	x	x	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x
Organization - Redefined	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Organization - Relative	x	x	c	x	x		x	c	x	c	x			x	x	x	x	x	x	x	x	x
Organization - Serial	x	x	c	x	x	c	x	c	x	c	x			x	x	x	x	x	x	x	x	x
Organization - SQL row	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Organization - User Interface			c			c		c						c	c	c	c	c	c	c	c	c

Table 9. Record elements (continued)

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
Organization - Working storage	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Prologue	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Record	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Record ID item	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Record length item	c	c	c	c	c	c	c	c	c	c				x	x	x	c	x	c	x	c	x
Record name	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Redefinition for	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
SQL table names	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Variable length item (DL/I)			x	x	x	x	x	x	x													x

Note: *Includes Windows 95 and Windows 98, Windows NT, and Windows 2000.

Legend: In this table, the following characters are used to indicate the level of support:
x Supported
c Supported with compatibility considerations
g The part or some of its features can be connected in a GUI application
blank Not supported

Alternate specification

"Alternate specification" is the name of an existing record whose data item structure is to be used for this record.

Alternate specification

Uses

Specify "Alternate specification" to avoid creating and maintaining several record structures. Once you change the structure of one record, you change the structure of all records that refer to it as an alternate specification.

There is no record structure defined for this record. VisualAge Generator uses the structure defined in the record named as the "Alternate specification".

Definition considerations for Alternate specification

For VSAMRS files accessed using VisualAge Generator Server for MVS, VSE, and VM or VisualAge Generator Server (specifying the VSAMRS file type at generation) or for VSAM files on CICS, the file name for the alternate specification cannot be the same as the file name for any other record in the program.

Indexed records

When used with indexed records, an alternate specification enables you to specify an alternate record ID item that can be associated with an alternate index. You must also specify an alternate file name.

SQL row records

When used with another SQL record, alternate specification records allow a program to do the following:

- Simultaneously scan two different sets of rows from the same relational table
- Access a table with a different default key item
- Access a table with different default selection conditions

You cannot specify an SQL row record as an alternate specification for a record with a different organization.

You cannot specify a record with a different organization as an alternate specification for an SQL row record.

You cannot enter SQL table names for a row defined as an alternate specification for another record.

Target environments for Alternate specification

Environment	Compatibility considerations
VM CMS	If you generate the program to use COBOL I/O statements to access a VSAM file using an alternate index (file type is specified as VSAM at generation), then all programs in the job step that do I/O to the same file must include both the alternate specification record and the base record defined with the primary key. Both records must specify the same file name. All programs must include both of the records, either as I/O objects or listed in the table and additional records list.
VM batch	Same as VM CMS.
CICS for MVS/ESA	For VSAM files, the file name for the alternate specification cannot be the same as the file name for any other record in the program.
MVS/TSO	For VSAMRS files accessed using VisualAge Generator Server for MVS, VSE, and VM or VisualAge Generator Server (specifying the VSAMRS file type at generation), the file name for the alternate specification cannot be the same as the file name for any other record in the program. If you generate the program to use COBOL I/O statements to access a VSAM file using an alternate index (file type is specified as VSAM at generation), then all programs in the job step that do I/O to the same file must include both the alternate specification record and the base record defined with the primary key. Both records must specify the same file name. All programs must include both of the records, either as I/O objects or listed in the table and additional records list.
MVS batch	Same as MVS/TSO.
IMS/VS	None.
IMS BMP	Same as MVS/TSO.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	Same as MVS/TSO.
CICS for OS/2	Same as CICS for MVS/ESA.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	Same as CICS for MVS/ESA.

Alternate specification

Environment	Compatibility considerations
Windows NT	None.
CICS for Windows NT	Same as CICS for MVS/ESA.
Solaris	None.
CICS for Solaris	Same as CICS for MVS/ESA.
Test Facility	None.

Default key item (SQL)

Default key item specifies the name of the data item to be used as the search field in default SQL statements built for an alternate specification of an SQL row record.

Uses

The default key item is used as the search field in the SQL statements that access records in relational databases.

Specifying the Default key item is optional, but is the only way to specify an SQL key item for an alternate specification for record.

Default key item is most useful for accessing tables that have a single column for which a unique index is defined. If a default key is specified, the key item cannot be modified by a REPLACE function.

If specified, the default key item must be in the item list for the primary record associated with the alternate specification record.

If you did not specify Alternate specification, you cannot specify Default key item.

Target environments for Default key item (SQL)

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.

Environment	Compatibility considerations
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Default selection conditions (SQL)

Default selection conditions are default search criteria specified for a record defined as an SQL row.

Uses

Default selection conditions refer to SQL search-conditions defined in conjunction with a record that is automatically included in default SELECT statements built for functions that access that record. You enter the search condition in the WHERE clause of a simulated SELECT statement.

Default selection conditions are useful for defining join conditions that need to be specified for an SQL row record that is defined as a join of two or more relational tables. If the SQL row record represents a single table with one or more columns combining to form a unique index, you would usually specify items in the record as key items instead of coding default selection conditions.

If an index is defined for any columns referenced in the search conditions, the item that represents the column in the SQL row record must be defined as read-only, or the index is not used in the search in an UPDATE or SETUPD function.

Definition considerations for Default selection conditions

Join conditions are search conditions that express the relationships between the combined tables. Join conditions limit the number of rows in the larger table by selecting only valid combinations of rows. If no join conditions are defined, all possible combinations of rows are formed.

Using the WHERE clause

Default selection conditions are specified using the syntax for a search condition in the WHERE clause of a SELECT statement.

The syntax is not validated until the SQL statements with the default selection conditions are preprocessed by the relational database manager.

The default selection conditions are built into any WHERE clauses generated for functions with the SQL row record as the I/O object.

If key items are also specified for the SQL row, the default selection conditions are combined with any default key selection conditions using an AND logical operator.

Target environments for Default selection conditions

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.

Environment	Compatibility considerations
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

File name

File name associates a record specification with a physical file.

Uses

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

File name is a 1- to 8-character file name that 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 special characters or embedded blanks or have an EZE prefix.
- The name cannot use $, #, or a COBOL reserved word if the file is associated with:
  - VSAM or SEQ in a non-CICS environment
  - OS2COBOL in a CICS OS/2 environment

Note: The $, #, and @ are not in the National Language syntactic character set and cannot be represented by equivalent code points across differing code pages. Avoid using these characters if the program you are developing will be exported or generated for these differing code pages. This will particularly affect programs exported between the System/370 host and the workstation.

File name

Definition considerations for File name

Records sharing the same file name are associated with the same physical file. The physical file associated with the file name can be specified during program generation. The default destination can be overridden during resource association or by using the EZEDEST special function word.

Programs that run together in the same run unit and access the same physical file must have the same file name specified for all records associated with the file.

All records with the same file name that run together in the same run unit must have the same attributes (record format, length, organization, key length, and key offset). They must also match the physical file definition.

If you define a message queue record, you must specify a file name.

Note: For more information on size restrictions and record lengths, see "Appendix C. Size restrictions and record lengths" on page 683.

Generation Considerations for File name

Using the linkage table, you can specify whether a file associated with the file name is at a remote location and whether automatic data conversion from ASCII to EBCDIC is to be performed when file records are accessed.

For more information on accessing remote files, refer to *VisualAge Generator Client/Server Communications Guide*.

Target environments for File name

Environment	Compatibility considerations
VM CMS	If the program has not set the EZEDEST special function word for the record, this file name is used as the file name on a CMS FILEDEF command or DLBL command to allocate the physical file prior to running the program.
VM batch	If the file is associated with a VSAM or VM file and the program has not set the EZEDEST special function word for the record, the value is the file name used when the file is opened.
CICS for MVS/ESA	The file name is the default system resource name. Its meaning is based on the file type selected when the program is generated or tested.
MVS/TSO	If the program has not set the EZEDEST special function word for the record, use this file name as the file name on a TSO ALLOC command or DLBL command to allocate the physical file prior to running the program.

Environment	Compatibility considerations
MVS batch	If the file is associated with a VSAM or sequential MVS file and the program has not set the EZEDEST special function word for the record, the value is the DD name used when the file is opened. If a serial file is associated with a GSAM file, the value is the DD name used in the JCL. It must match the DD name specified in the GSAM DBD.
IMS/VS	The file name is the default logical terminal or transaction code used when records are added to a serial output file allocated to an IMS message queue. The actual logical terminal or transaction code used when the program is running must be defined to IMS. File name is ignored for an input message queue because it uses the I/O PCB.
IMS BMP	If the file is associated with a VSAM or sequential MVS file, and the program has not set the EZEDEST special function word for the record, the value is the DD name used when the file is opened. If a serial output file is associated with a message queue, the value is the default logical terminal or transaction code. The actual logical terminal or transaction code used when the program is running must be defined to IMS. File name is ignored for an input message queue because it uses the I/O PCB. If a serial file is associated with a GSAM file, the value is the DD name used in the JCL. It must match the DD name specified in the GSAM DBD.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	If the program has not set the EZEDEST special function word for the record, the file name is the DLBL name used when the file is opened. Only the first 7 characters of the name are used.
CICS for OS/2	Same as CICS for MVS/ESA except that use with message queue records is not supported.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	Same as CICS for MVS/ESA.

File name

Environment	Compatibility considerations
Windows NT	None.
CICS for Windows NT	Same as CICS for MVS/ESA.
Solaris	None.
CICS for Solaris	Same as CICS for MVS/ESA.
Test Facility	The test facility resolves the logical filename specified in the record to the physical filename. The Resource Association File (RAF) is used to connect the logical file name to the physical file name at test time.

Key item (DL/I)

Key item specifies the name of an item in a DL/I segment record that contains the segment key.

Uses

The default value is blank. Use the default value if the DL/I segment has no sequence field.

Definition considerations for Key item

The Key item must have the same name, length, and offset that the segment sequence field has in the DL/I database description.

Target environments for Key item

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	Not supported.

Environment	Compatibility considerations
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
CICS for AIX	Not supported.
Windows NT	Not supported.
CICS for Windows NT	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
Test Facility	None.

Number of occurrences item

Number of occurrences item supports the definition of variable-length records for files in which all records have a fixed-length part at the beginning of the record, followed by an array with a variable number of entries at the end of the record.

Uses

Number of occurrences item is valid only for indexed, message queue or serial records.

If the record you define ends with an array that can have a variable number of occurrences, specify the name of the data item that contains the number as the Number of occurrences item.

The data item that contains the number of occurrences must meet all of the following requirements:

- Be defined in the fixed-length part of the variable-length record
- Have a data type of numeric (Num), binary (Bin), or packed (Pack)
- Have a maximum length of 9 characters
- Contain no decimal places

Definition considerations for Number of occurrences item

The number of occurrences item contains the number of entries in the array. When the record is written to the file, VisualAge Generator computes the length of the record by multiplying the current value in the number of occurrences item by the length of an array entry.

Number of occurrences item

For message queue records, the occurrences item value multiplied by the length of an array item plus the length of the record structure without the array determines the message length. If the record contains both a record length item and occurrences item, the record length item is set to the length calculated from the number of occurrences before a message is added to the queue.

The array is represented in the record data item definition by the last item that is not subordinate to any other item (not part of a substructure). The array itself can be substructured. The dimension (occurrences) specified for the array is the maximum number of entries that can be written out for the record. The minimum number of entries is zero.

The number of occurrences item must not be specified for fixed-length records.

If the records in a file are variable-length, the record specification must include a record length item, a number of occurrences item, or both.

If you have both a record length item and a number of occurrences item, the record length is calculated using the number of occurrences item each time the record is written to the file. The calculated length is moved to the record length item before writing the record.

Test and run-time use the number of occurrences item only when reading records from the file or writing records to the file.

Program statements can reference all the items in the entire record regardless of the values of the record length item and number of occurrences item.

Target environments for Number of occurrences item

Environment	Compatibility considerations
VM CMS	Variable-length records are supported in VSAM files and VM sequential files. Variable-length records in VM non-VSAM sequential files have a 4-byte header (2-byte length field and 2-byte filler field). The data item definition for the record should not include the header. The 4-byte variable length header is added when writing to a VM non-VSAM sequential file with variable record format and removed when the record is read.
VM batch	Same as VM CMS.
CICS for MVS/ESA	Variable-length records are not supported for temporary storage queues and transient data queues.

Environment	Compatibility considerations
MVS/TSO	Variable-length records are supported in VSAM files and MVS sequential files. Variable-length records in MVS non-VSAM sequential files have a 4-byte header (2-byte length field and 2-byte filler field). The data item definition for the record should not include the header. The 4-byte variable length header is added when writing to an MVS non-VSAM sequential file with variable record format and removed when the record is read.
MVS batch	If the file is a GSAM file, the 2-byte leading length field is added to records written to the file and removed from records read from the file. The data item definition for the record should not include the header. If the file is not a GSAM file, then the same considerations for MVS/TSO apply to MVS batch.
IMS/VS	Number of occurrences item is only supported for serial files associated with IMS message queues. The IMS message header (length, ZZ field, and transaction code) is added to records written to a queue and removed from records read from the queue. The data item definition for the record should not include the header.
IMS BMP	If the file is a GSAM file, the 2-byte leading length field is removed from records read from the file and added to records written to the file. The data item definition for the record should not include the header. For files associated with IMS message queues, the considerations for IMS/VS apply to IMS BMP. Otherwise, the considerations for MVS/TSO apply to IMS BMP.
CICS for VSE/ESA	VSE supports variable length sequential files. Variable-length records in VSE non-VSAM sequential (SAM) files have an eight-byte header. The first four bytes are the block length descriptor (BL) and the next four bytes are the record length (RL) descriptor. The value in BL includes the length of both BL plus RL. The value in RL includes the length of RL. In both the BL and RL, bytes 0 through 1 are the length in binary format. Bytes 2 through 3 are reserved. This is true for both variable length blocked and unblocked records. Variable unblocked records have a blocking factor of one.
VSE batch	Same as CICS for VSE/ESA.
CICS for OS/2	Variable-length records are supported with CICS-managed files (generation file type VSAM), and COBOL-managed files (file type OS2COBOL). Variable-length records are not supported for temporary storage queues and transient data queues. Use with message queue records is not supported.

Environment	Compatibility considerations
OS/400	Not supported.
OS/2 (GUI)	A program can be considered to be a part to a GUI program. Therefore, the program or features of the program can be connected in a GUI program.
Windows (GUI)	Same as OS/2 (GUI).
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	Variable length records are supported for CICS-managed files with file type VSAM in the resource association file.
Windows NT	None.
CICS for Windows NT	Same as CICS for AIX.
Solaris	None.
CICS for Solaris	Variable length records are supported for CICS-managed files with file type VSAM in the resource association file.
Test Facility	None.

Organization

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

Uses

The following are the types of organization supported:

- DL/I segment
- Indexed
- Message queue
- Redefined
- Relative
- Serial
- SQL row
- Working storage
- User interface

Record specification options vary depending on the record organization you specify.

Target environments for Organization

Support for an organization means that I/O operations can be performed for the record in the environment.

Except for redefined records, all types of records can be included in GUI programs.

All types of records can be used in any other type of program as temporary storage data structures by specifying the record name in the called parameter list or the table and additional records list.

Organization - DL/I segment

DL/I segment organization indicates that the record is a segment in a DL/I database.

Uses

The record name must be the same as the segment name in the DL/I database.

Target environments for DL/I segment

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.

Organization - DL/I segment

Environment	Compatibility considerations
CICS for AIX	Not supported.
Windows NT	Not supported.
CICS for Windows NT	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
Test Facility	None.

Organization - Indexed

Indexed organization indicates that the records are in a file and are accessed by a key.

Uses

The record key is specified in the record ID item.

Target environments for Indexed

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	Not supported.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.

Environment	Compatibility considerations
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Organization - Message queue

A message queue record represents a message on an MQSeries message queue. Message attributes include instructions for processing the message. I/O operations on message queues are like I/O operations on serial files. Only the ADD, SCAN and CLOSE I/O options are supported for message queue records.

Message queue record definitions include the following message queue record attributes:

Include message in transaction
Includes the message as a recoverable resource in the program's unit of work.

Open queue for exclusive use on input
Opens the message queue for exclusive use on input. If this attribute is not specified, the queue is opened for shared use.

The following message queue record attributes specify records used as parameters on MQ API calls. If a record is not specified for an attribute, a default record is built based on the attributes specified for the message queue record.

Queue descriptor record
MQ Object Descriptor, MQOD, record.

MQOD is used as a parameter on MQSeries MQOPEN and MQCLOSE calls to functions that open and close queues.

Open options record
MQ Open Options, MQOO, record.

MQOO is used as a parameter on MQSeries MQOPEN and MQCLOSE calls to functions that open and close queues.

Message descriptor record
MQ Message Descriptor, MQMD, record.

MQMD is used as a parameter on MQSeries MQGET and MQPUT calls to functions that implement the ADD and SCAN I/O options for message queue records.

Get options record
MQ Get Message Options, MQGMO, record.

MQGMO is used as a parameter on the MQSeries MQGET call to the function that implements the SCAN I/O option for a message queue record.

Put options record
MQ Put Message Options, MQPMO, record.

MQPMO is used as a parameter on the MQSeries MQPUT call to the function that implements the ADD I/O option for a message queue record.

Message queue record definitions also include the following record attributes:

- File name
- Alternate specification
- Record length item
- Occurrences item

Definition considerations for Message queue

A message queue record can be defined as a unit of related data items (data structure), similar to the definition of other record organizations. One or more single, unrelated data items can be defined for use in the message queue record instead of, or in addition to, the data structure.

Message queue records provide the following:

- Data items to temporarily hold message data
- Data items to be passed as arguments to another program

Target environments for Message queue

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.

Environment	Compatibility considerations
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	Not supported.
OS/400	None.
OS/2 (GUI)	None.
Windows (GUI)	None.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Organization - Redefined

A redefined record is an alternate data item structure for an existing record.

Uses

The alternate data structure enables you to access the data in a record using different data item names and definitions.

You cannot use redefined records as I/O objects, but you can use them in statements and as passed parameters.

To use a redefined record, you must specify the name of the record in the Tables and Additional Record List during program definition. The record that it redefines must also be referenced in the program as an I/O object, additional record, or working storage record.

Organization - Redefined

Target environments for Redefined

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Organization - Relative

Relative organization indicates that the file is an ordered set of fixed-length records accessed by a relative number.

Uses

The relative number is found in the record ID item specified for the record.

For relative records, the record ID item does not need to be part of the record structure. It can be an item in any map, record, or table used in the program.

Target environments for Relative

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	When serial or relative files are associated with temporary storage queues, an additional byte is added to the front of the record. The VisualAge Generator record definition should not include this byte. However, non-VisualAge Generator programs sharing the same temporary storage queue must allocate space for the byte and maintain its value. A zero (0) in the additional byte means the record logically exists in the file. A one (1) in the additional byte means it has been deleted. The record length for a deleted record should be 1. Functions operating on temporary storage queues have the following actions: • An ADD function sets this byte to '0'. • A DELETE function sets this byte to '1' and sets the record length to 1. • An INQUIRY function for a record with a value of '1' in the additional byte causes the NRF record state to be set. • A REPLACE function sets this byte to '0'. • A SCAN function skips records with a byte value of '1' in the first byte space. • An UPDATE function for a record with a byte value of '1' in the additional byte causes the NRF record state to be set.
MVS/TSO	None.
MVS batch	None.
IMS/VS	Not supported.
IMS BMP	None.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	None.
CICS for OS/2	Same as CICS for MVS/ESA.
OS/400	Not supported.
OS/2 (GUI)	None.
Windows (GUI)	None.
OS/2 (C++)	None.

Organization - Relative

Environment	Compatibility considerations
AIX	None.
HP-UX	None.
CICS for AIX	Same as CICS for MVS/ESA.
Windows NT	None.
CICS for Windows NT	Same as CICS for MVS/ESA.
Solaris	None.
CICS for Solaris	Same as CICS for MVS/ESA.
Test Facility	None.

Organization - Serial

Serial organization indicates that the records are stored in the file in sequential order.

Uses

References to the records start at the beginning and go consecutively to the end of the file.

With serial files, you can only use the ADD, SCAN, or CLOSE function options.

Target environments for Serial

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	When serial or relative files are associated with temporary storage queues, an additional byte is added to the front of the record. The VisualAge Generator record definition should not include this byte. However, non-VisualAge Generator programs sharing the same temporary storage queue must allocate space for the byte and maintain its value. A zero (0) in the additional byte means the record logically exists in the file. A one (1) in the additional byte means it has been deleted. The record length for a deleted record should be 1. Functions operating on temporary storage queues have the following actions: • An ADD function sets this byte to '0'. • A SCAN function skips records with a byte value of '1' in the first byte space.

Environment	Compatibility considerations
MVS/TSO	None.
MVS batch	None.
IMS/VS	Transaction programs cannot use a serial file for input. Batch programs can only use one serial file for input.
IMS BMP	None.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	None.
CICS for OS/2	Same as CICS for MVS/ESA.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	Same as CICS for MVS/ESA.
Windows NT	None.
CICS for Windows NT	Same as CICS for MVS/ESA.
Solaris	None.
CICS for Solaris	Same as CICS for MVS/ESA.
Test Facility	None.

Organization - SQL row

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

Target environments for SQL row

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.

Organization - SQL row

Environment	Compatibility considerations
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Organization - User interface

A User Interface (UI) record is a special type of record you can use in Web transaction programs. UI records are generated into HTML pages. Data items, tables and literals in a UI record are generated into HTML parts included in the page.

These records specify how data is defined and processed but not how it is displayed. User Interface Records (UI records) can be the object of CONVERSE and XFER with UI record. This programming model separates the concerns of the business logic developer from the user interface developer. The outputs of UI record generation are a Java bean, which encapsulates all the defined processing of the business data and a Java Server Page, which accesses this bean. These outputs are deployed on the web server and the Java Server Page. An HTML expert can then complete the user interface without having to consider how the run-time data populates the page.

Definition considerations for User interface

Data items for this type of record can be laid out freely as in a Working Storage Record with the addition of User Interface related record attributes and record item attributes. For information on defining UI record data item edits, see "UI record data item edits" on page 246. For information on defining data item UI types, see "Data item UI type" on page 231. UI Record attributes are:

UI title
Default title for the UI record

Submit value item
Name of the data item in the UI record that will receive a value as defined by any of the items that have a UIType of Submit or Submit Bypass. Because HTML pages provide no predefined way to capture the values of function keys pressed by end users (no EZEAID equivalent), this item along with the Submit and Submit Bypass items are used to provide this function. Submit and Submit Bypass items set up all the possible values available to the end user and the Submit Value Item receives the value of the key selected by the end user. Define this item if the values to be checked in the Web Transaction program must be more descriptive than 'PF1', 'Enter', and the like. If you do not specify a value, the default behavior is to check PF values and use EZEAID. In this case, the values for the Submit and Submit Bypass items must be strings like 'PF1' as outlined in the following list.

The submit value item must be defined in the UI record's data item list with a Char, Mixed, DBCS, or Unicode data type. This item may be an array item. An array item is an item defined with an occurs value greater than one.

The submit value item must not be an occurrences or selected index item defined for a data item in the UI record's data item list.

By default, the Submit value item field is blank and EZEAID is the defined submit value item. Only the following string values are valid with the default definition:

- 'PF1' - 'PF24'
- 'PA1' - 'PA3'
- 'ENTER'

'ENTER' is used if the specified value is not valid.

Input edit order
Input edits are processed at run time according to the input edit order.

The default input edit order is set as input items are created in the UI record, from the top of the record to the bottom of the record. You can change the input edit order.

Help text
Default help text for the entire UI Record.

UI record default HTML generation

The generation of default HTML both during ITF execution and JavaServer Pages (JSP) generation is defined through a combination of UI record data item attributes (type, length, occurs, and so on), substructuring, UIType, UIType Properties and Edits. Table 10 and the sections which follow describe how these elements are combined for creating different HTML elements and default layout. For most elements, if the item is occurred it simply means repeat the element for as many occurs as there are. However, for some combinations an occurred item will cause different HTML elements to be generated.

Table 10 gives a basic description of how different HTML elements are related to the UI record item definition.

Table 10. HTML elements and UI record item definition

HTML element(s)	Item attributes	Occurs	UI type	UI type properties	Edits	Notes
Text Input	N/A	1	Input, Input/Output	N/A	N/A	N/A
Checkbox	CHA, Numeric	1	Input, Input/Output	N/A	Boolean	value = 'Y' or 'N' for CHA value = 0 or 1 for Numeric
TextArea	CHA, MIX, Length>80	1	Input, Input/Output	N/A	N/A	N/A
Plain Text	N/A	1	Output	N/A	N/A	N/A
Plain Text Paragraph	N/A	>1	Output	Selected Index Item is NOT defined.	N/A	Each array element will be a separate line in the paragraph.

Table 10. HTML elements and UI record item definition (continued)

HTML element(s)	Item attributes	Occurs	UI type	UI type properties	Edits	Notes
Submit Button	CHA, MIX	1	Submit, Submit Bypass	N/A	N/A	Buttons only show if there are values in the item. Default values can be set in UIType Properties.
HyperText Link	N/A	1	Program Link	Link properties define what will be used for the **HREF** attribute of the **<A>** HTML element.	N/A	Parameters defined in the Link Properties are defined as query parameters on the URL generated for the **HREF** attribute of the **<A>** HTML element.
Form	N/A	1	Form	Link properties define what will be used for the **ACTION** attribute of the **<FORM>** HTML element.	N/A	Parameters defined in the Link Properties are defined as Hidden Input fields within the **<FORM>** HTML element.
Drop Down List	N/A	1	Input, Input/Output	No Selected Index Item defined	Match Valid Edit Table	The data of this list is constant based on the data in the table. The selected value will be contained in the item that references the Edit Table.

Table 10. HTML elements and UI record item definition (continued)

HTML element(s)	Item attributes	Occurs	UI type	UI type properties	Edits	Notes
Drop Down List	N/A	>1	Output	Selected Index Item is defined.	N/A	The list is composed of the data contained in the item at run time. The actual index of the item selected is set into the defined Selected Index Item. If the index item is occurred the list will be multiple select.
Table	N/A	>1	Output	Selected Index Item can be optionally defined	N/A	Substructed items at the next level define the columns. The labels of these items will be the column headers. These items can be further substructured to give structure to the cells of the column. If a Selected Index Item is defined the first column of the table will contain radio buttons or checkboxes for handling single or multiple select respectively.

Selected Index Item: This item must be a numeric item. If it is occurred, the list will be a multiple select list. The number of occurrences of the Selected Index Item must be the same as the item referencing it in this case.

In the case where the list item is substructured under an item of UIType=Form, the selected index item is of little value. This is because a Form defines the invocation of an independent program. This means that one cannot index into an existing set of data because the program is invoked anew each time and this set of data does not exist. This is unlike returning back from a CONVERSE in which the existing set of data still exists and can readily be indexed using the Selected Index Item values.

Occurrences item: The value in this referenced item determines how many occurrences should show in the list. If no item is referenced, then all elements in the array will show.

Labels: In most cases, the label defined for an item is displayed in bold next to or above the appropriate HTML element. However, there are some special cases to note:

- For an occurred item (either explicitly defined or implicitly as result of parent item being occurred) one can define a single label for all occurrences or a label for each occurrence. Each line of the label definition is a separate label. Define one label to have the same label for all occurrences. To have a separate label for each occurrence, define as many labels (one on each line) as there are occurrences.
- If no label is defined for a Program Link, Submit, or Submit Bypass item, the data of the item itself is used as the label.

HTML element layout: The following areas of HTML element layout can be controlled by the UI record developer:

- In general, HTML elements will show up in the order that the items have been defined in the UI record. There are a few exceptions:
  - Submit/Submit Bypass items that are NOT substructured will show up across the bottom of the page no matter where they are defined relative to the other items.
  - In HTML, Forms cannot contain Forms. Because the UI Record itself implicitly defines a Form, items with UIType=Form cannot be interspersed with items that are meant to be in the default form. To handle this situation, the default generation will place the Form elements after all the other elements regardless of where they were defined in the record.
- In general, items at the top structure level create line breaks between their generated HTML elements. However, if an item is substructured the sub item HTML elements will flow from left to right without line breaks. The exception to this rule are items with UIType=Form. Basically the flow rule

described above starts over with items substructured under a Form item. The highest level items under a Form item will create line breaks between themselves. To get items to flow from left to right, substructure them further under another item. Note: if the only purpose of the super item is for layout purposes use a filler item. An item named * is a filler item.

Target environments for User interface

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only for Web transaction programs.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only for Web transaction programs.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only for Web transaction programs.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	Valid only for Web transaction programs.
AIX	Valid only for Web transaction programs.
HP-UX	Valid only for Web transaction programs.
CICS for AIX	Valid only for Web transaction programs.
Windows NT	Valid only for Web transaction programs.
CICS for Windows NT	Valid only for Web transaction programs.
Solaris	Valid only for Web transaction programs.
CICS for Solaris	Valid only for Web transaction programs.
Test Facility	Valid only for Web transaction programs.

Organization - Working storage

Working storage records define storage areas for temporary data items that are used in programs.

Uses

The data item values are not saved when the program has finished running unless the data items have been moved to a record and placed in a file.

Definition considerations for Working storage

A working storage record can be defined as a unit of related data items (data structure), similar to the definition of other record organizations. One or more single, unrelated data items can be defined for use in the working storage record instead of, or in addition to, the data structure.

If an input message to a main transaction consists of only the transaction name followed by blanks, the program assumes it is being started with no working storage record being passed.

Working storage records provide the following:

- Data items to temporarily hold data, such as the date or intermediate results of calculations
- Data items to be passed as arguments to another program

Level-77 data items

Single data items are referred to as level-77 data items. These data items are defined with a level of 77 after all data items in the working storage structure have been defined.

Level-77 data items are initialized to blanks, or to numeric or binary zeros, depending on the defined data type.

Level-77 items are included in a program only if the working storage record is specified as the primary working storage record in the program specification.

If a working storage record is passed as a parameter to another program, only the structure is passed. Any level-77 data items you want to pass must be specified as separate arguments on the CALL statement.

Generation Considerations for Working storage

The primary working storage record identified in the program specification is always initialized. If the /INITADDWS generation option is specified, working storage records included in the Table and Additional Record List are initialized based on the type of data (blanks for character, DBCS, Unicode, and mixed data, and zero for numeric data).

Target environments for Working storage

Environment	Compatibility considerations
VM CMS	None.

Environment	Compatibility considerations
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Working storage records are included in a GUI program by specifying the working storage record name as the name of a record part dropped on the GUI definition. Level-77 items are included with each record.
Windows (GUI)	Same as OS/2 (GUI).
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Prologue

A prologue is a text description of the record.

Uses

The use of the prologue is optional. The prologue area is used for documentation purposes only. It is for commentary and does not affect the program at run time.

Target environments for Prologue

Supported in all environments without compatibility considerations.

Record

A record or multiple records are individually accessible units of storage in a file or database.

Uses

Records can also be used as temporary working storage when a program runs. A record definition consists of the following:

- A specific record organization

  The record organization indicates both the structure of the file or database containing the collection of records and how to gain access to the record. You can choose from the following types of organization:
  - DL/I segment
  - Indexed
  - Redefined
  - Relative
  - Serial
  - SQL row
  - User interface
  - Working storage
- A list of data items

Target environments for Record

Supported in all environments without compatibility considerations.

Record ID item

Record ID item is the name of the data item that contains the record key for an indexed file, or the relative record number for a relative file.

Definition considerations for Record ID item

Indexed records

For an indexed file, the record ID item must be defined in the Data items list for the record.

The record ID item should have the same length and record offset as the key in the records in the physical file.

Relative records

The record ID item does not have to be specified in the Data items list as part of the record structure for a relative record.

The item should be defined as follows:

Data type
Numeric (Num), packed (Pack), or binary (Bin)

Decimal places
0

Maximum length
9

If you have not defined the record ID item anywhere in your program and implicits are allowed for the program, test and generation defines a 2-byte binary implicit data item.

When a relative record file is accessed while the program is running, the record ID item must contain a number that indicates the record position in the file relative to the beginning of the file.

Target environments for Record ID item

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.

Environment	Compatibility considerations
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Record length item

Record length item specifies the data item that contains the length of a variable-length serial, indexed or message queue record.

Uses

When a variable-length record is read from the file, the length is stored in the record length item. When a record is written to the file, the length is obtained from the record length item. The data item specified as the Record length item does not have to be defined in the record definition itself.

The data item characteristics of the record length item must be one of the following:

- Have a data type of numeric (Num), binary (Bin), or packed, (Pack)
- Have a maximum length of 9 digits
- Contain no decimal places

Definition considerations for Record length item

If the record length item is not defined anywhere in the program and implicits are allowed in the program, the record length item is defined implicitly as a 2-byte binary field.

The maximum length for a variable-length record is the record length calculated from the data item definition for the record. When a variable-length record with a record length item is written to the file, the value in the record length item must be less than or equal to the maximum record length.

For message queue records, the message length is set equal to the value in the record length item when a program adds the record to a message queue. When the program reads a message from the queue, the message length is returned in the record length item. If the record contains both a record length item and occurrences item, the record length item is set to the length calculated from the number of occurrences before a message is added to the queue.

The record length item must not be specified for fixed-length records.

Record length item

If the records in a file are variable-length, the record specification must include a record length item, a number of occurrences item, or both.

If you have both a record length item and a number of occurrences item, the record length is calculated using the number of occurrences item each time the record is written to the file. The calculated length is moved to the record length item before writing the record.

Test and run-time use the record length only when reading records from the file or writing records to the file.

Program statements can reference all the items in the entire record regardless of the values of the record length item and number of occurrences item.

For platform specific record lenghts, see "Maximum record lengths" on page 684.

Target environments for Record length item

Environment	Compatibility considerations
VM CMS	Variable-length records are supported in VSAM files and VM sequential files. Variable-length records in VM non-VSAM sequential files have a 4-byte header (2-byte length field and 2-byte filler field). The data item definition for the record should not include the header. The 4-byte variable length header is added when writing to a VM non-VSAM sequential file with variable record format and removed when the record is read.
VM batch	Same as VM CMS.
CICS for MVS/ESA	Variable-length records are not supported for temporary storage queues and transient data queues.
MVS/TSO	Variable-length records are supported in VSAM files and MVS sequential files. Variable-length records in MVS non-VSAM sequential files have a 4-byte header (2-byte length field and 2-byte filler field). The data item definition for the record should not include the header. The 4-byte variable length header is added when writing to an MVS non-VSAM sequential file with variable record format and removed when the record is read.
MVS batch	If the file is a GSAM file, the 2-byte leading length field is removed from records read from the file and added to records written to the file. The data item definition for the record should not include the header. Otherwise, the considerations for MVS/TSO apply to MVS batch.

Environment	Compatibility considerations
IMS/VS	Record length item is only supported for serial files associated with IMS message queues. The IMS message header (length, ZZ field, and transaction code) is removed from records read from a queue and added to records written to the queue. The data item definition for the record should not include the header.
IMS BMP	If the file is a GSAM file, the 2-byte leading length field is removed from records read from the file and added to records written to the file. The data item definition for the record should not include the header. For files associated with IMS message queues the same considerations for IMS/VS apply to IMS BMP. Otherwise, the considerations for MVS/TSO apply to IMS BMP.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	Same as CICS for MVS/ESA.
CICS for OS/2	Variable-length records are supported with CICS-managed files (generation file type VSAM), and COBOL-managed files (file type OS2COBOL). Variable-length records are not supported for temporary storage queues and transient data queues. Use with message queue records is not supported.
OS/400	Not supported.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	Variable length records are supported for CICS-managed files with file type VSAM in the resource association file.
Windows NT	None.
CICS for Windows NT	Same as CICS for AIX.
Solaris	None.
CICS for Solaris	Variable length records are supported for CICS-managed files with file type VSAM in the resource association file.
Test Facility	None.

Record name

The record name identifies a record part.

Uses

See Appendix B. Naming conventions for data item, record, function names for record naming conventions.

Target environments for Record name

Supported in all environments without compatibility considerations.

Record data structure

Record data structure, called a record data item definition, specifies a data structure that describes the format or field layout of data items within a record.

Uses

The structure definition is specified as a list of data items. The following elements can be specified for each data item in the list:

- BYTES
- DEC (decimal positions)
- DESCRIPTION
- LENGTH
- LEVEL
- NAME
- OCCURS
- TYPE
- USAGE

These data items are discussed in "Data item" on page 206.

Target environments for Record data structure

See the individual data item discussions in "Data item" on page 206.

Redefinition for

The redefinition for element identifies the name of the record that is being redefined when record organization is specified as redefined record.

Uses

You cannot use redefined records as I/O objects, but you can use them in statements and as passed parameters.

To use a redefined record, you must specify the name of the record in the Table and Additional Record List during program definition.

Target environments for Redefinition for

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

SQL row record data structure

SQL row record data structure defines a set of data items that represent columns in the SQL tables represented by the SQL row.

Uses

The structure definition is specified as a list of data items. The following elements can be specified for each data item in the list:

- BYTES

- DEC (decimal positions)
- DESCRIPTION
- KEY
- LENGTH
- NAME
- READ-ONLY
- SQL COLUMN NAME
- SQL DATA CODE
- TYPE
- USAGE

These data items are discussed in "Data item" on page 206.

Target environments for SQL row record data structure

See the individual data item discussions in "Data item" on page 206.

SQL table names

SQL table names is the set of relational tables that an SQL row record represents.

Uses

Two elements are specified for each table: table name and table label. The data that you enter for the table name and table label must meet SQL naming conventions. SQL names are not validated.

SQL Table Name

SQL table name is the name of a table or view as it is known by the database manager. The name can be in any format that is accepted by the database manager. You can qualify the name with an owner or authorization identifier and a remote database location, or you can specify a synonym for a table name instead of the table name itself. The table name can be specified directly (as a literal) or indirectly (as a host variable). If you use host variable format, the program must move the table or view name to the variable at run time.

Literal (direct): The value of table name must meet SQL naming conventions. VisualAge Generator does not validate SQL names.

The name is left-justified and padded with blanks and is not changed to uppercase. The name is inserted in SQL statements and passed to the database manager exactly as it is entered.

Host variable (indirect): When using host variable format, the table name is a data item name preceded by the SQL host variable indicator. The program must move the SQL table name to the data item at run time. The SQL table names in host variable format can be as follows:

- The SQL host variable indicator is defined in the environment variable EZERSQLHOST. The default character can be changed by your system administrator using the customization procedures for language-dependent options. You can also change the default character by using the EZERSQLHOST environment variable.

  Note: For VisualAge Generator Developer, all support for EZER* environment variables have been removed. The SQL host variable indicator is specified on the VAGen-SQL tab in the VisualAge Preferences dialog. Some of these environment variables are still supported by VisualAge Generator Server.
- A valid data item name preceded by a question mark (?) if you want to specify a table name in SQL host variable format in the external source format file.
- The data item must define a CHA or MIXED data item.
- The data item name can be qualified, subscripted, or both. The subscript can be a numeric data item or a literal. The table name has a maximum length of 60 characters, including the SQL host variable indicator.

At run time, the program must move the actual table name into the host variable before the SQL record is accessed. The SQL statements for functions that gain access to the record as an I/O object are prepared and executed dynamically when the program runs. The current contents of the data item for the host variable are substituted wherever it appears in the SQL statement.

The value moved into the table name host variable must meet SQL naming conventions. The generated program does not validate the value.

SQL table label

SQL table label is a shortened version of the table name. You can enter a label up to 4 characters long. The label is used 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.

If you do not specify the label, record specification automatically generates one for you.

Table Joins

If there is more than one table specified, the record represents a join of the tables in the list. When a record represents a join, use the Default Selection Conditions element to specify default join conditions to limit the number of rows produced when the tables are joined. SQL rows that represent joins cannot be used with I/O options that modify the database (UPDATE, SETUPD, REPLACE, DELETE, and ADD).

Target environments for SQL table names

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable length item (DL/I)

Variable length item is the name of a data item in a DL/I segment that contains the length of the rest of the segment, including the length field.

Uses

The name must be specified if the segment has variable length. The item must be a 2-byte binary item and have the same length and offset as the length field in the segment in the DL/I database description.

If the segment has fixed length, the variable-length item field must be left blank.

The variable-length item will usually be the first item in the segment. The only time the variable-length item is not first is when a concatenated segment in a logical database is built from a fixed-length segment followed by a variable-length segment.

Target environments for variable length item (DL/I)

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
CICS for AIX	Not supported.
Windows NT	Not supported.
CICS for Windows NT	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
Test Facility	None.

Chapter 5. Tables

A table part is an array of predefined data values that can be used for the following:

- Editing data that is entered on a map by a user of the program (edit table types)
- Storing messages that the program issues (message table type)
- Storing information for reference by a program when it runs (all table types)

Tables are included in a program by specifying the table name as the program message table, in the table and additional records list for the program, or as the name of a table part included in a GUI definition. The table that a program uses for its message table must be identified in the message table prefix for the program.

Table elements

Table 11. Table elements

Element	COBOL											GUI		C++								
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	Test Facility
Column definition	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Contents definition	x	x	x	x	x	x	x	x	x	c	x	x	x	x	x	x	c	x	x	x	c	x
Prologue	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Resident	c	c	c	c	c	c	c	c	c	c	c	x	x	c	c	c	c	c	c	c	c	x
Shared	c	c	c	c	c	c	c	c	c	c	x	c	c	c	c	c	c	c	c	c	c	c
Table name	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Table type	x	x	x	x	x	x	x	x	x	x	c	c	c	x	x	x	x	x	x	x	x	c

Note: *Includes Windows 95 and Windows 98, Windows NT, and Windows 2000.

Table 11. Table elements (continued)

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	

Legend: In this table, the following characters are used to indicate the level of support:
x Supported
c Supported with compatibility considerations
blank Not supported

Column definition

A table column definition specifies a data structure that describes the format of each row in a generator data table as opposed to a relational database table.

Uses

The column definition is specified as a list of data items. The following item elements can be specified for each data item in the list:

- BYTES
- DEC (decimal positions)
- DESCRIPTION
- LENGTH
- LEVEL
- NAME
- TYPE
- USAGE

These data items are discussed in "Data item" on page 206.

Target environments for Column definition

Supported in all environments without compatibility considerations.

Contents definition

A table contents definition defines the contents of the data values stored in the table.

Uses

The contents are specified in a list. Each row in the list represents one row in the table. Each row is formatted as specified in the table column definition.

For tables, the comparison is always done from the first row in the table through the last row in the table. For better performance, values that occur most frequently should be put first in the table.

Table contents are converted to the format of the runtime environment when the table is generated for an environment.

Target environments for Contents definition

Supported in all environments without compatibility considerations.

Prologue

The prologue area is used for documentation purposes only.

Uses

The use of a prologue is optional; it is commentary only and does not affect program execution.

Target environments for Prologue

Supported in all environments without compatibility considerations.

Resident

Resident keeps the shared table in storage after its use count is set to zero.

Uses

A use count is kept for shared tables accessed by programs. If Resident is specified, the table contents are not removed from memory when the use count is set to zero.

Note: You can only specify Resident for tables that are shared.

Definition considerations for Resident

The time it takes to delete the table contents depends on the target runtime environment.

The Keep after use attribute determines when the use count is incremented and decremented for a specific program.

Target environments for Resident

Environment	Compatibility Considerations
VM CMS	Resident table contents are removed from storage when any of the following occurs: • The main program ends • The main program performs an XFER • The main program performs a DXFR to a non-VisualAge Generator program • A called program returns to a non-VisualAge Generator program.
VM batch	Same as VM CMS.
CICS for MVS/ESA	Do not use the RES keyword on the CICS PPT entry for the table program if you use the resident attribute. Resident table contents are deleted only when the CICS region comes down or when the VisualAge Generator Server for MVS, VSE, and VM new copy utility is used to delete the table. When new copy is used, transactions that are already active and have the table loaded continue to use the old copy of the table until the table use count is set to zero for the transaction. The old copy is deleted when no more programs are using the copy.
MVS/TSO	Same as VM CMS.
MVS batch	Same as VM CMS.
IMS/VS	Resident table contents are deleted when any of the following occurs: • A main batch program ends • A called program returns to a non-VisualAge Generator program • A main transaction program finishes processing all messages in the input message queue
IMS BMP	Same as VM CMS.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	Same as VM CMS.
CICS for OS/2	Same as CICS for MVS/ESA.
OS/400	Same as VM CMS.
OS/2 (GUI)	None.
Windows (GUI)	None.
OS/2 (C++)	Resident table contents are deleted when the run unit ends.
AIX	Same as OS/2 (C++).
HP-UX	Same as OS/2 (C++).
CICS for AIX	Same as OS/2 (C++).

Environment	Compatibility Considerations
Windows NT	Same as OS/2 (C++).
CICS for Windows NT	Same as OS/2 (C++).
Solaris	Same as OS/2 (C++).
CICS for Solaris	Same as OS/2 (C++).
Test Facility	None.

Shared

Shared specifies whether all users of a table share a single copy of the table.

Uses

If you do not specify Shared, each program user has a unique copy of the table.

Note: You can only specify Resident for tables that are shared.

Target environments for Shared

Environment	Compatibility Considerations
VM CMS	Shared tables marked as resident remain loaded until any of the following ends VisualAge Generator Server for MVS, VSE, and VM: • Return to the system or the non-VisualAge Generator program • XFER • DXFR to non-VisualAge Generator program Otherwise, the shared indicator is ignored. Each user has a separate copy of the table.
VM batch	Same as VM CMS.
CICS for MVS/ESA	If shared tables are modified, the modifications are effective for all users of the table in the same CICS region until the table is reloaded. If the table being modified has synchronization considerations with other transactions using the table, the modifications to the table should not be made across a CALL statement or an I/O option. Programs requiring synchronization across CALL statements or I/O options should use an external serialization method.
MVS/TSO	Same as VM CMS.
MVS batch	Same as VM CMS.
IMS/VS	Shared tables cannot be modified by the program.
IMS BMP	Same as VM CMS.

Environment	Compatibility Considerations
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	Same as VM CMS.
CICS for OS/2	If shared tables are modified, the modifications are effective for all users of the table on the same CICS for OS/2 system until the table is reloaded. Updates to shared tables are not synchronized in CICS for OS/2. Programs requiring synchronization for updates to shared tables should use an external serialization method. An example of an external serialization method would be to call an external non-VisualAge Generator program to perform a CICS ENQ function while the table is being updated.
OS/400	None.
OS/2 (GUI)	All GUIs running under the same IBM image in a single function share the same copy of a shared table. If the table is not marked as shared, each GUI has a separate copy of the table. GUIs do not share tables with called programs.
Windows (GUI)	Same as OS/2 (GUI).
OS/2 (C++)	Shared tables cannot be updated.
AIX	Shared tables cannot be updated.
HP-UX	Shared tables cannot be updated.
CICS for AIX	Shared tables cannot be updated.
Windows NT	Shared tables cannot be updated.
CICS for Windows NT	Shared tables cannot be updated.
Solaris	Shared tables cannot be updated.
CICS for Solaris	Shared tables cannot be updated.
Test Facility	All GUIs running under the test facility share the same copy of a shared table. If the table is not marked as shared, each GUI has a separate copy of the table. All other types of programs running under the test facility share the same copy of a table, whether or not it is marked as shared. Programs running under the test facility do not share tables with generated programs that are called from the programs being tested.

Table name

The table name is the name of the table part.

Definition considerations for Table name

Naming conventions for tables:

Maximum length
7

First character
Alphabetic (A-Z)

Other characters
Alphanumeric (A-Z, 0-9)

DBCS name
No

- Table names cannot end with a 0 (zero)
- The table name must be unique within a CICS execution system and within a target MVS load library
- To avoid potential conflicts with the program names generated for the map groups, do not end the table name with FM or P1
- Table names cannot begin with the EZE prefix.
- Table names cannot contain embedded blanks.
- Table names cannot be COBOL reserved words.

Other rules apply for message tables. The format of the message table name is *xxxxyyy* where *xxxx* is the message table prefix and *yyy* is a suffix that identifies the national language. The format for the message table name prefix follows.

- Maximum length: 4
- First character: alphabetic (A-Z)
- Other characters: alphanumeric (A-Z, 0-9)

The message table prefix is specified during program specification.

A suffix is appended to the message table prefix to build the name of the user message table. The VisualAge Generator Developer supports the following suffixes for the national languages:

Code	Language
CHS	Simplified Chinese
CHT	Traditional Chinese
DES	Swiss German
DEU	German
ENP	Uppercase English
ENU	US English
ESP	Spanish

Table name

FRA French
ITA Italian
JPN Japanese
KOR Korean
PTB Brazilian Portuguese

Note: Uppercase English is not supported by AIX, OS/2, Windows NT, HP-UX, SCO OpenServer, and Solaris.

Target environments for Table name

Supported in all environments without compatibility considerations.

Table type

Table type defines how the table is to be used.

Uses

You can specify the following table types:

Unspecified
Defines a table to store information to which a program refers when it runs.

Statements can refer to a table with the Unspecified type.

You cannot use a table with the Unspecified type as an edit routine.

Match valid
Defines a table that requires the data entered by a program user to match a value in the first column of the table.

A match valid table can be specified as a map variable field edit routine. This type of table is useful for checking a set of valid entries for a map field.

Match invalid
Defines a table that requires that the data entered by a program user does not match any of the data in the first column of the table.

A match invalid table can be specified as a map variable field edit routine.

Range match valid
Defines a table that requires the data entered by a program user to be between sets of values.

A range match valid table can be specified as a map variable field edit routine. It must have at least two columns, with the first and second columns showing the valid ranges.

When a map variable field has a range valid table specified as an edit table, each time a value is entered in the field it is checked against each row of the table to see if it is greater than or equal to the first column, and less than or equal to the second column. If the range check fails, the value is treated as not valid. If the range check passes, the value is treated as valid.

Message

Defines a table to contain user messages for your program to use.

Message tables must have at least two columns, the first column contains the message number and the second column contains the message text.

Program messages are used to notify the user of errors detected in validated input from maps. You identify which message is to be displayed in response to an edit error by coding the program to move a message number to EZEMNO or by specifying a message number as the Edit Error Message Number.

When an error is detected, the text from the second column for the selected error message is displayed in map field EZEMSG. The second column can be defined to be longer than 78 characters, but if it is longer than the field defined using the EZEMSG special function word, the value will be truncated.

The message table columns must follow the following conventions:

The first two columns of a message table with the lowest level data item must meet the following requirements:

Column 1

This data item is used for the message number.

Type Num

Length

4

Decimals

0

Column 2

This data item is used for the message text.

Type Char or Mixed

Length

1 to 254 (78 is recommended)

Decimals

0

Target environments for Table type

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	MESSAGE type tables may not be referenced in a statement. Only access to MESSAGE tables is via EZEMNO or as a message number reference for a map edit failure.
OS/2 (GUI)	Functions related to table type are not supported in GUIs. All tables are treated as if the table type was unspecified.
Windows (GUI)	Same as OS/2 (GUI).
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	Results can vary for range checks in different environments due to differences in data formats in the environment.

Chapter 6. Items

An item is a data element. The item can be defined by itself as a separate part, or within the context of a data structure definition.

Item elements

Table 12. Item elements

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
Data item bytes	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Data item decimal places	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Data item description	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Data item key (SQL row record)	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Data item length	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Data item level	x	x	x	x	x	x	x	x	x	c	x	c	c	c	x	x	x	c	c	x	x	c
Data item name	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Data item occurs	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Data item read-only (SQL row record)	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x

Table 12. Item elements (continued)

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
Data item SQL column (SQL row record)	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Data item SQL data code (SQL row record)	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Data item type - Bin	x	x	x	x	x	x	x	x	x	c	x	c	c	c	x	x	x	c	c	x	x	c
Data item type - Char	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
Data item type - DBCS	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
Data item type - Hex	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Data item type - Mixed	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
Data item type - Num	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
Data item type - Numc	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
Data item type - Pacf	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
Data item type - Pack	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
Data item type - Unicode												x	x	x	x	x	x	x	x	x	x	x
Data item UI type			c			c		c						c	c	c	c	c	c	c	c	c

Table 12. Item elements (continued)

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
Data item UI type - Form			c			c		c						c	c	c	c	c	c	c	c	c
Data item UI type - Hidden			c			c		c						c	c	c	c	c	c	c	c	c
Data item UI type - Input			c			c		c						c	c	c	c	c	c	c	c	c
Data item UI type - Input/Output			c			c		c						c	c	c	c	c	c	c	c	c
Data item UI type - None			c			c		c						c	c	c	c	c	c	c	c	c
Data item UI type - Output			c			c		c						c	c	c	c	c	c	c	c	c
Data item UI type - Program link			c			c		c						c	c	c	c	c	c	c	c	c
Data item UI type - Submit			c			c		c						c	c	c	c	c	c	c	c	c
Data item UI type - Submit bypass			c			c		c						c	c	c	c	c	c	c	c	c
Data item usage	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
UI record data item edits			c			c		c						c	c	c	c	c	c	c	c	c

Table 12. Item elements (continued)

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
UI record data item edits - Check SO/SI space			c			c		c						c	c	c	c	c	c	c	c	c
UI record data item edits - Currency			c			c		c						c	c	c	c	c	c	c	c	c
UI record data item edits - Currency symbol			c			c		c						c	c	c	c	c	c	c	c	c
UI record data item edits - Edit function			c			c		c						c	c	c	c	c	c	c	c	c
UI record data item edits - Edit table			c			c		c						c	c	c	c	c	c	c	c	c
UI record data item edits - Edit type			c			c		c						c	c	c	c	c	c	c	c	c
UI record data item edits - Fill character			c			c		c						c	c	c	c	c	c	c	c	c
UI record data item edits - Fold			c			c		c						c	c	c	c	c	c	c	c	c

Table 12. Item elements (continued)

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
UI record data item edits - Input required			c			c		c						c	c	c	c	c	c	c	c	c
UI record data item edits - Maximum value			c			c		c						c	c	c	c	c	c	c	c	c
UI record data item edits - Minimum input			c			c		c						c	c	c	c	c	c	c	c	c
UI record data item edits - Minimum value			c			c		c						c	c	c	c	c	c	c	c	c
UI record data item edits - Numeric Separator			c			c		c						c	c	c	c	c	c	c	c	c
UI record data item edits - Run edit function on web			c			c		c						c	c	c	c	c	c	c	c	c
UI record data item edits - Sign			c			c		c						c	c	c	c	c	c	c	c	c

Table 12. Item elements (continued)

Element	COBOL											GUI		C++								
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	Test Facility
UI record data item edits - Zero edit			c			c		c						c	c	c	c	c	c	c	c	c

Note: *Includes Windows 95 and Windows 98, Windows NT, and Windows 2000.

Legend: In this table, the following characters are used to indicate the level of support:
x Supported
c Supported with compatibility considerations
blank Not supported

Data item

An item is a data element. The item can be defined by itself as a separate part, or within the context of a data structure definition.

Shared data item definitions are stored independently of the records or tables to which they belong.

This section describes the language elements that specify data item characteristics. Some characteristics can be stored in a data item part independent of any record or table structure if the usage of the data item definition is shared. Other elements specify information about a data item as it is used in that structure.

If the usage of the data item definition is specified as shared, the following data item characteristics are independent of the structure:

BYTES
DEC (Decimal positions)
DESCRIPTION
LENGTH
NAME
TYPE

If the usage of the data item definition is specified as nonshared, the characteristics above are stored with the data structure, not with a separate data item.

A data item name represents the same data item wherever it is used as a shared data item in a record or table. Changing the shared data item specifications of type, length, and decimal places in one structure causes a corresponding change in all structures that include the data item as a shared data item. Changing the characteristics of a nonshared data item does not affect any other data item definition.

The following data item characteristics are always stored with the data structure, treated as local data and dependent on the structure:

KEY (SQL row record)
LEVEL
OCCURS
READ-ONLY (SQL row record)
SQL COLUMN NAME (SQL row record)
SQL DATA CODE (SQL row record)
USAGE

Data item bytes

Data item bytes specify the number of bytes required to store the data item internally.

Uses

If data item length is specified, the bytes are automatically calculated from the length value.

Maximum Number of Characters

The following table explains the maximum number of characters for records and tables, based on the type of data:

Data type	Length in record	Bytes in record	Length in table	Bytes in table
Char	32767	32767	254	254
Mixed	32767	32767	254	254
DBCS	16383	32766	127	254
UNICODE	16383	32766	127	254
Hex	65534	32767	254	127

Maximum Number of Digits

Data item bytes

The following table explains the maximum number of digits for records and tables, based on the type of data:

Data type	Length in record	Bytes in record	Length in table	Bytes in table
Num/Numc	18	18	18	18
Pack/Pacf	18	10	18	10
Bin	18	8	18	8
SQL Bin	9	4	N/A	N/A
SQL Pack	18	10	N/A	N/A

For binary data, the following table shows the correspondence between the number of digits and the number of bytes required:

Length (in digits)	Bytes
1 - 4	2
5 - 9	4
10 - 18	8

Target environments for Data item bytes

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.

Environment	Compatibility considerations
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Data item decimal places

Data item decimal places specifies the number of places reserved to the right of an implied decimal point. The default is 0 (no decimal places).

Uses

The length of the data item must include space for the decimal places. The maximum number of decimal places is 18 or the length of the data item, if it is shorter than 18. The decimal point is not stored with the data.

You cannot specify Decimal places when either of the following conditions is true:

- You select a data type that is not numeric.
- For SQL data items, you select a data type other than PACK.

Target environments for Data item decimal places

Supported in all environments without compatibility considerations.

Data item description

Data item description is a text description of what the data item represents.

Uses

You can specify a 1- to 30-character description of a data item.

Descriptions can be entered in uppercase, lowercase, or mixed case.

Target environments for Data item description

Supported in all environments without compatibility considerations.

Data item key

Data item key designates whether a data item in an SQL row record is a key column in an SQL row.

Uses

Columns that are defined as key items are used as the selection variables when building the default SQL statements for INQUIRY, UPDATE, SETINQ, and SETUPD functions.

Key items are not replaced by the default SQL statement built for the REPLACE I/O option.

Any column designated as a key is included in the ORDER BY clause of the SELECT statement for a SETINQ function. The order of the key within the record definition determines its position within the ORDER BY clause.

The data item key designation is ignored if the data item definition is used with an alternate specification record. Use the default key item to specify a key column for an alternate specification record.

Target environments for Data item key

Supported in all environments without compatibility considerations.

Data item length

Data item length specifies the number of characters or digits set aside in a data structure for a single occurrence of a data item.

The bytes set aside for an array is length times occurs.

Uses

If the Bytes value is specified, the Length value is automatically calculated from the bytes value.

Maximum Number of Characters

The following table explains the maximum number of characters for records and tables, based on the type of data:

Data type	Length in record	Bytes in record	Length in table	Bytes in table
Char	32767	32767	254	254
Mixed	32767	32767	254	254
DBCS	16383	32766	127	254
UNICODE	16383	32766	127	254

Hex	65534	32767	254	127

Maximum Number of Digits

The following table explains the maximum number of digits for records and tables, based on the type of data:

Data type	Length in record	Bytes in record	Length in table	Bytes in table
Num/Numc	18	18	18	18
Pack/Pacf	18	10	18	10
Bin	18	8	18	8
SQL Bin	9	4	N/A	N/A
SQL Pack	18	10	N/A	N/A

For binary data, the following table shows the correspondence between the number of digits and the number of bytes required:

Length (in digits)	Bytes
1 - 4	2
5 - 9	4
10 - 18	8

Target environments for Data item length

Supported in all environments without compatibility considerations.

Data item level

Data item level specifies a number that can be used to create a substructure within the data items in a record or table.

Uses

Level information is unique to a data structure definition. Level numbers can differ for the same data item that is used in several data structures. The only valid levels are 3 through 49, and 77.

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 byte length of data items in a substructure must be equal to the length of the data item at the next higher level in the structure. The default level number is 3.

Definition considerations for Data item level

A data structure can contain one or more filler data items (nonshared data items with a * specified for the name). The length of the filler data item must be included in the entire length of a structure.

Working storage records can contain single data items in addition to or in place of a data structure. Level 77 data items are not part of a data structure. If both a data structure and single data items are defined, the structure must be defined first. The level 77 data items will follow the structured data items.

Level 77 data items can be used for relative record ID items, work items, or arguments to be passed to another program in a CALL statement. They are not passed as part of the working storage when the working-storage record is passed as a parameter on CALL, XFER, or DXFR statements. They are not included if the working-storage record is specified in the Table and Additional Record List for the program.

Target environments for Data item level

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	Binary items are stored in byte-reversed order in OS/2 and Windows environments. Therefore, using levels to access bytes within binary items will have different results than in other environments.
OS/400	None.
OS/2 (GUI)	Same as CICS for OS/2.
Windows (GUI)	Same as CICS for OS/2.

Environment	Compatibility considerations
OS/2 (C++)	Same as CICS for OS/2.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	Same as CICS for OS/2.
CICS for Windows NT	Same as CICS for OS/2.
Solaris	None.
CICS for Solaris	None.
Test Facility	Same as CICS for OS/2.

Examples for Data item level

The following example depicts how various lengths of data entered in a certain field level must total the next higher field level.

Name	Level Numbers	Length
ADDRESS	10	42
STREET	12	20
LOCALE	12	22
CITY	14	10
STATE	14	2
NEWZIP	14	10
ZIP	16	5
*	16	5

Notice in the table above that the total length of the data items in a substructure must equal the length of the owning data item. For example:

- ADDRESS is level 10 and has a length of 42. STREET and LOCALE are level 12 and have a combined length equal to ADDRESS.
- CITY, STATE, and NEWZIP are level 14 and have a combined length equal to LOCALE.
- ZIP and * (a filler) are level 16 and have a combined length equal to NEWZIP.

Data item name

Data item name is the unique identification of a data item within a data structure.

Uses

For data item naming conventions, see Appendix B. Naming conventions for data item, record, function names.

Target environments for Data item name

Supported in all environments without compatibility considerations.

Data item occurs

Data item occurs specifies the number of occurrences of the data item, allowing the creation of arrays within a record.

Uses

A number from 1 to 32767 can be specified. The default is 1.

Definition considerations for Data item occurs

If you define a data item with more than 1 occurrence, no other data items within its substructure can have more than 1 occurrence.

The same data item can be used in other data structures. The occurs characteristic for a data item applies only for the data structure where the characteristic is defined. The data item can have a different number of occurrences for each data structure in which it is used.

For information on handling occurs in GUI programs, refer to the *VisualAge Smalltalk User's Guide*.

You cannot define level-77 data items in working storage with an occurrence greater than 1.

You cannot define the number of occurrences for an item in an SQL row record. An SQL row is treated as a set of single data items at the same level.

You cannot define the number of occurrences for an item in a table. For a table item, the number of occurrences is equal to the number of rows defined for table contents.

Target environments for Data item occurs

Supported in all environments without compatibility considerations.

Data item Read-only

Data item read-only prevents the data item from being written to the relational database.

Uses

The specification of read-only determines what columns are included in the generated SQL statements that write to the relational database.

Definition considerations for Data item Read-only

Specify Read-only for columns from a view that you know cannot be updated and for columns that your program never needs to change.

Read-only is automatically specified when the following is true:
- Data items in SQL column names are expressions.
- Data items in an SQL row record are defined as an SQL join.

Target environments for Data item Read-only

Supported in all environments without compatibility considerations.

Data item usage

Data item usage indicates whether the data item definition is stored as a separate data item or stored as part of the data structure definition.

Uses

Data item usage can be set to the following:
- Nonshared
- Shared

Nonshared characteristics apply only to the definition of the item in that data structure and the characteristics are stored with the part containing the data structure.

Shared characteristics apply wherever a shared item with the same name is defined in any data structure. Shared characteristics are stored in a data item part, independent of the data structures, function local storage lists, or function parameter lists to which they belong.

Map fields are always nonshared.

Data item usage applies only to where VisualAge Generator stores and retrieves the information about the data item, not to the usage of the item in terms of generated code.

Definition considerations for Data item usage

A data item name represents the same data whenever it is used as a shared data item in a data structure. Changing the shared data item specifications of

type, length, and decimal places in one structure causes a corresponding change in all structures that include the data item as a shared data item. Changing the characteristics of a nonshared data item does not affect any other data item definition.

When you export an ENVY application containing a new shared data item, both a VisualAge Generator shared data item and a shared data element are created. VisualAge Generator Developer creates the shared data element with the same name as the shared data item.

Nonshared data item characteristics are saved with the containing data structure. Nonshared data item information is not saved as a separate data item part.

Target environments for Data item usage

Supported in all environments without compatibility considerations.

Data item SQL column name

Data item SQL column name specifies the column name used in the relational database. The name can be from 1- to 36-characters.

Uses

If you do not enter a name, the data item name is used as the SQL column name.

Definition considerations for Data item SQL column name

The SQL column can be the name of a column in a relational table or view definition, or an SQL expression made up of column names, SQL operators, constants, and built-in functions.

Relational table column

Specify a relational table column name if the actual name of the column in the relational table or view definition differs from the data item name. If the SQL row was defined as a join of multiple tables or views, the column name should be qualified by the table label to which it belongs. The table label is defined for the table or view name in the record specification to which it belongs.

SQL expression

Specify an SQL expression to define a virtual column that can be used as a read-only data item in the SQL row definition. The expression can be made up of column names, SQL operators, constants, and built-in functions. The expression is calculated when the SQL row is read from the database.

An example of an expression used as a column name is as follows:

```
MONTHLY-SALARY * 12
```

The name is inserted into the generated SQL statements just as it is entered. All single-byte characters not within double quotes are folded to uppercase. The specified name is not validated by VisualAge Generator. Instead, the name is checked by the relational database manager during SQL statement preparation for a program.

Target environments for Data item SQL column name

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Data item SQL data code

Data item SQL data code is the number that identifies the data type of the data item in the relational database.

Data item SQL data code

Uses

SQL data codes can vary only for DBCS, character, hexadecimal, and unicode data items. SQL data codes are fixed for other types of data items and cannot be modified.

SQL data codes are set correctly if you retrieved the data item definition for the record from the relational database. If you enter the data item definitions, specify the SQL data code to be the same as the SQL data code defined for the associated column in the database for the data item.

When specified for hexadecimal data items, the SQL data code lets a program access SQL data types not supported by corresponding VisualAge Generator data types. For example, to access a double-precision FLOAT column in a relational database, define the corresponding data item with a type of hexadecimal, a bytes value of 8, and an SQL data code of 481.

Valid combinations of SQL data codes, and character or DBCS data items are as follows:

Table 13. SQL Data Types for Variable and Fixed Length Columns

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

For more information on SQL data codes, refer to *VisualAge Generator Design Guide*

Target environments for Data item SQL data code

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.

Environment	Compatibility considerations
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Data item type

Data item type specifies the internal format or type of data. The data type determines how the item is processed when referenced in processing statements.

Uses

The following types of data are available:

Bin Binary number
CHA Character data
DBCS Double-byte character data
Hex Hexadecimal data
Mixed DBCS data mixed 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
Double-byte character data

Performance Information for numeric data types

VisualAge Generator supports five numeric data types to allow you to define record structures that match the structure of records in existing files.

For new data item definitions, in general, use BIN because it requires the least amount of storage and performs the best overall.

For zoned decimal data, NUMC performs better than NUM. For packed decimal data, PACK performs better than PACF.

BIN data type is the most efficient for array subscripting and relative record IDs. Try to use short binary positive numbers with no decimal places. "Short" includes numbers whose values are less than 32768 (or defined as four numeric digits), which can be resolved into a length of two bytes.

NUM data without decimal places is more efficient in calculations, moves, and comparisons than numeric data with decimals. If decimal places are required, the number of decimal places across all items in a calculation should be consistent.

VisualAge Generator handles numeric or binary data with up to 18 digits, including decimal places. Performance is improved, however, if the fields contain 4 or less digits (including decimal places).

Target environments for Data item type

See the individual data item types for compatibility considerations.

Data item type - Bin

Bin (binary) specifies numeric data stored in binary format.

Uses

Binary data can store large numbers in a smaller number of bytes than other numeric data types.

Target environments for Data item type - Bin

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.

Environment	Compatibility considerations
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	Binary items are stored in byte-reverse order on OS/2 and Windows systems. Therefore, using substructures for binary items can have different results that in other systems.
OS/400	None.
OS/2 (GUI)	Same as CICS for OS/2.
Windows (GUI)	Same as CICS for OS/2.
OS/2 (C++)	Same as CICS for OS/2.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	Same as CICS for OS/2.
CICS for Windows NT	Same as CICS for OS/2.
Solaris	None.
CICS for Solaris	None.
Test Facility	The test facility stores binary numbers in INTEL format in byte reversed order.

Data item type - CHA

CHA (character) data consists of alphabetic, numeric, or national characters.

Target environments for Data item type - CHA

ASCII character sets are used in workstation environments. EBCDIC character sets are used in host environments. Differences in collating sequence can cause greater-than or less-than comparisons to have different results in ASCII environments than in EBCDIC environments.

Environment	Compatibility considerations
VM CMS	Uses EBCDIC character sets.
VM batch	Uses EBCDIC character sets.
CICS for MVS/ESA	Uses EBCDIC character sets.
MVS/TSO	Uses EBCDIC character sets.
MVS batch	Uses EBCDIC character sets.
IMS/VS	Uses EBCDIC character sets.
IMS BMP	Uses EBCDIC character sets.
CICS for VSE/ESA	Uses EBCDIC character sets.
VSE batch	Uses EBCDIC character sets.
CICS for OS/2	Uses ASCII character sets.
OS/400	Uses EBCDIC character sets.
OS/2 (GUI)	Uses ASCII character sets.
Windows (GUI)	Uses ASCII character sets.
OS/2 (C++)	Uses ASCII character sets.
AIX	Uses ASCII character sets.
HP-UX	Uses ASCII character sets.
CICS for AIX	Uses ASCII character sets.
Windows NT	Uses ASCII character sets.
CICS for Windows NT	Uses ASCII character sets.
Solaris	Uses ASCII character sets.
CICS for Solaris	Uses ASCII character sets.
Test Facility	Uses ASCII character sets.

Data item type - DBCS

DBCS (double-byte character set) data consists of double-byte characters. DBCS data is ideographic character data that requires two positions for each character in a record, table, or map.

Uses

DBCS data requires a terminal or printer device with DBCS capability so that double-byte character data can be viewed or printed. Double-byte characters are required for languages such as Japanese, Korean, and Chinese.

Target environments for Data item type - DBCS

ASCII character sets are used in workstation environments. EBCDIC character sets are used in host environments. Differences in collating sequence can cause greater-than or less-than comparisons to have different results in ASCII environments than in EBCDIC environments.

Environment	Compatibility considerations
VM CMS	Uses EBCDIC character sets.
VM batch	Uses EBCDIC character sets.
CICS for MVS/ESA	Uses EBCDIC character sets.
MVS/TSO	Uses EBCDIC character sets.
MVS batch	Uses EBCDIC character sets.
IMS/VS	Uses EBCDIC character sets.
IMS BMP	Uses EBCDIC character sets.
CICS for VSE/ESA	Uses EBCDIC character sets.
VSE batch	Uses EBCDIC character sets.
CICS for OS/2	Uses ASCII character sets.
OS/400	Uses EBCDIC character sets.
OS/2 (GUI)	Uses ASCII character sets.
Windows (GUI)	Uses ASCII character sets.
OS/2 (C++)	Uses ASCII character sets.
AIX	Uses ASCII character sets.
HP-UX	Uses ASCII character sets.
CICS for AIX	Uses ASCII character sets.
Windows NT	Uses ASCII character sets.
CICS for Windows NT	Uses ASCII character sets.
Solaris	Uses ASCII character sets.
CICS for Solaris	Uses ASCII character sets.
Test Facility	Uses ASCII character sets.

Data item type - Hex

HEX data consists of bytes of data where each byte is represented by two hexadecimal (base 16) digits.

Data item type - Hex

Uses

Hexadecimal data items provide basic processing functions (moves, comparisons and parameter passing) for database fields whose data type is not directly supported by VisualAge Generator.

If the data type of the data item in a physical record does not match any of the other data types, select HEX. Specify a Bytes value equal to the number of bytes the field uses in the database record or file.

The Length value represents the number of digits in the hexadecimal data item and is twice the Bytes value.

Hexadecimal data items cannot be used in arithmetic expressions.

Target environments for Data item type - Hex

Supported in all environments without compatibility considerations.

Data item type - Mixed

Mixed data can contain both single-byte (SBCS) and double-byte characters (DBCS).

Uses

Mixed data requires a terminal or printer device with DBCS capability so that double-byte character data can be viewed or printed. Double-byte characters are required for languages such as Japanese, Korean, and Chinese.

The length specified for a mixed data item type is the number of single-byte characters that the field can contain. The number of Bytes for a mixed field must equal the length.

Relational database managers do not support a data type for mixed data. Instead, they allow mixed data in character columns when the database is running in a DBCS environment. When accessing mixed data in a relational database, the character items in the SQL row record must be defined as character data items instead of mixed data items. The character data items can be moved to mixed variable fields on maps or in other data structures.

Definition considerations for Data item type - Mixed

On systems that use EBCDIC character sets (mainframes like the System/370 and AS/400), special delimiters identify DBCS subfields within a mixed data item. The shift-out (SO) character in SBCS text signifies that the text following the SO character is DBCS. The shift-in (SI) character in DBCS text signifies that the text following the SI character is SBCS. If you are defining mixed data items for records that will be stored on mainframes, ensure that the item length includes space for SO/SI characters for all valid values for the item.

Target environments for Data item type - Mixed

ASCII character sets are used in workstation environments. EBCDIC character sets are used in host environments. Differences in collating sequence can cause greater-than or less-than comparisons to have different results in ASCII environments than in EBCDIC environments.

Environment	Compatibility considerations
VM CMS	Uses EBCDIC character sets.
VM batch	Uses EBCDIC character sets.
CICS for MVS/ESA	Uses EBCDIC character sets.
MVS/TSO	Uses EBCDIC character sets.
MVS batch	Uses EBCDIC character sets.
IMS/VS	Uses EBCDIC character sets.
IMS BMP	Uses EBCDIC character sets.
CICS for VSE/ESA	Uses EBCDIC character sets.
VSE batch	Uses EBCDIC character sets.
CICS for OS/2	Uses ASCII character sets.
OS/400	Uses EBCDIC character sets.
OS/2 (GUI)	Uses ASCII character sets.
Windows (GUI)	Uses ASCII character sets.
OS/2 (C++)	Uses ASCII character sets.
AIX	Uses ASCII character sets.
HP-UX	Uses ASCII character sets.
CICS for AIX	Uses ASCII character sets.
Windows NT	Uses ASCII character sets.
CICS for Windows NT	Uses ASCII character sets.
Solaris	Uses ASCII character sets.
CICS for Solaris	Uses ASCII character sets.
Test Facility	Uses ASCII character sets.

Data item type - Num

NUM data is numeric data in character (or zoned decimal) format.

Data item type - Num

Uses

NUM is supported for compatibility with previous products. For new development, use BIN or PACK for defining numeric data items.

Definition considerations for Data item type - Num

Internally, each digit is represented by the character for that digit. The data value is right-justified padded on the left with character zeros. The sign of the number is stored in the left half of the last byte (the zone).

In EBCDIC, a positive sign is represented by the standard zone value for a numeric character, which is hexadecimal F. The negative sign is hexadecimal D.

A negative sign is represented by the hexadecimal digit 7.

NUM is not supported in relational databases.

Target environments for Data item type - Num

Environment	Compatibility considerations
VM CMS	EBCDIC sign formats are used.
VM batch	EBCDIC sign formats are used.
CICS for MVS/ESA	EBCDIC sign formats are used.
MVS/TSO	EBCDIC sign formats are used.
MVS batch	EBCDIC sign formats are used.
IMS/VS	EBCDIC sign formats are used.
IMS BMP	EBCDIC sign formats are used.
CICS for VSE/ESA	EBCDIC sign formats are used.
VSE batch	EBCDIC sign formats are used.
CICS for OS/2	ASCII sign formats are used.
OS/400	EBCDIC sign formats are used. Refer to the *VisualAge Generator Design Guide* and *VisualAge Generator Generation Guide* documents for more information on compatibility. You can optimize run-time performance by using the Generation option /POSSIGN=F.
OS/2 (GUI)	ASCII sign formats are used.
Windows (GUI)	ASCII sign formats are used.
OS/2 (C++)	ASCII sign formats are used.
AIX	ASCII sign formats are used.

Environment	Compatibility considerations
HP-UX	ASCII sign formats are used.
CICS for AIX	ASCII sign formats are used.
Windows NT	ASCII sign formats are used.
CICS for Windows NT	ASCII sign formats are used.
Solaris	ASCII sign formats are used.
CICS for Solaris	ASCII sign formats are used.
Test Facility	ASCII sign formats are used.

Data item type - Numc

NUMC data is numeric data in character (or zoned decimal) format with a system sign value.

Uses

NUMC is supported for compatibility with previous products. For new development, use BIN or PACK for defining numeric data items.

Definition considerations for Data item type - Numc

Internally, each digit is represented by the character for that digit. The data value is stored right-justified padded on the left with character zeros. The sign of the number is stored in the left half of the last byte (the zone).

In EBCDIC, NUMC data items are equivalent to NUM data items, except that the hexadecimal digit C represents a positive sign. The negative sign is hexadecimal D.

A negative sign is represented by the hexadecimal digit 7.

Select NUMC when a program creates records to be processed by other products using the C convention for positive sign.

NUMC is not supported in relational databases.

Target environments for Data item type - Numc

Environment	Compatibility considerations
VM CMS	EBCDIC sign formats are used.
VM batch	EBCDIC sign formats are used.
CICS for MVS/ESA	EBCDIC sign formats are used.

Data item type - Numc

Environment	Compatibility considerations
MVS/TSO	EBCDIC sign formats are used.
MVS batch	EBCDIC sign formats are used.
IMS/VS	EBCDIC sign formats are used.
IMS BMP	EBCDIC sign formats are used.
CICS for VSE/ESA	EBCDIC sign formats are used.
VSE batch	EBCDIC sign formats are used.
CICS for OS/2	ASCII sign formats are used.
OS/400	EBCDIC sign formats are used. Refer to the *VisualAge Generator Design Guide* and *VisualAge Generator Generation Guide* documents for more information on compatibility. You can optimize run-time performance by using the Generation option /POSSIGN=C.
OS/2 (GUI)	ASCII sign formats are used.
Windows (GUI)	ASCII sign formats are used.
OS/2 (C++)	ASCII sign formats are used.
AIX	ASCII sign formats are used.
HP-UX	ASCII sign formats are used.
CICS for AIX	ASCII sign formats are used.
Windows NT	ASCII sign formats are used.
CICS for Windows NT	ASCII sign formats are used.
Solaris	ASCII sign formats are used.
CICS for Solaris	ASCII sign formats are used.
Test Facility	ASCII sign formats are used.

Data item type - Pacf

PACF data items specify packed decimal data. Packed decimal data has 2 digits in every byte, with the sign in the right half of the last byte.

Uses

PACF is supported for compatibility with previous products. Use BIN or PACK data types for new development.

Definition considerations for data item type - Pacf

Internally, the data value is stored right-justified padded on the left with zeros. The positive sign is a hexadecimal F. The negative sign is hexadecimal D. B is accepted as a negative sign in data created using other products.

PACF is not supported in relational databases.

Target environments for Data item type - Pacf

Refer to the *VisualAge Generator Design Guide* and *VisualAge Generator Generation Guide* documents for more information on compatibility. You can optimize run-time performance for the OS/400 environment by using the generation option /POSSIGN=F.

Data item type - Pack

PACK data items specify packed decimal data. Packed decimal data has 2 digits in every byte, with the system generated sign value in the right half of the last byte.

Uses

Use PACK for decimal numbers (non-integer numbers) for programs that will normally run on MVS, VSE, VM, or OS/400. If the program is to be used regularly on workstations, use BIN for numeric data. Always use BIN for integer data.

Definition considerations for Data item type - Pack

Internally the data value is stored right-justified padded on the left with zeros. The positive sign is represented by hexadecimal C. The negative by hexadecimal D. B is accepted as a negative sign in data created using other products.

Target environments for data item type - Pack

Refer to the *VisualAge Generator Design Guide* and *VisualAge Generator Generation Guide* documents for more information on compatibility. You can optimize run-time performance for the OS/400 environment by using the generation option /POSSIGN=C.

Data item type - Unicode

Unicode is a 16 bit (2-byte) character encoding standard established by the Unicode Consortium. It's goal is to support all characters from all languages in one character set. In version 2.0 of the standard, the character set contains over 38,000 distinct coded characters from 25 supported scripts.

Java uses the Unicode character encoding for character strings within Java programs. However, since very few systems have Unicode keyboards, fonts,

or printers, Java converts strings between Unicode and the locale character set when displaying or printing data or reading data from the keyboard.

Data items with the Unicode data type are assumed to contain double byte Unicode characters.

Use the Unicode data type for better performance for 4GL parts used only within Java clients, and for storing text information for applications where the text can be entered in different languages.

Definition considerations for Data item type - Unicode

The Unicode data type is only available for VisualAge Generator Developer on Java.

Unicode items can be defined in records and tables, but not maps. Unicode data is entered and displayed from Java client programs or Web programs.

Unicode support is not available for 3270 maps.

Servers can store Unicode data directly in files or using the GRAPHIC/VARGRAPHIC SQL data type on UDB databases where Unicode has been specified as the code page for GRAPHIC data.

Unicode items can only be assigned, moved, or compared to other Unicode items. All comparisons are logical comparisons between the bit values of the items in Unicode.

Unicode items are padded with Unicode blanks when required. String functions operate on Unicode items as byte strings.

The length for a Unitcode item is expressed as the number of Unicode characters. The number of bytes in the item is twice the length.

Unicode literals are not supported. Use tables to define initialized Unicode variables.

Table contents for Unicode variables are entered as single byte or mixed character data. Table definition converts the character strings to the corresponding Unicode values based on the current locale.

Target environments for Data item type - Unicode

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.

Environment	Compatibility considerations
CICS for MVS/ESA	Not supported.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Not supported.
IMS BMP	Not supported.
CICS for VSE/ESA	Not supported.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2 (GUI)	None.
Windows (GUI)	None.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None
CICS for Solaris	None.
Test Facility	None.

Data item UI type

Data item UI type is available only for items defined in User Interface (UI) records.

The user interface type defines how the item is used in the user interface and together with other data item attributes such as occurs, substructuring, etc. help determine the default UI elements used to implement the HTML form when the UI record is generated.

Uses

The following Data item UI types are available:

Form Form is similar to Program Link except that it allows items that are substructured under an item with this UI Type to be part of an HTML

"Form". With the Form UI Type you can substructure items of UI Type=Input, Input/Output, Submit, etc. to create multiple Forms that invoke independent programs, passing the data of the given form.

Hidden
Fields of this type are not visible to the program user on the generated HTML page. However, the data of these fields will be passed when a form containing a hidden field is submitted.

Input Defines that input can be entered by the program user and that edits will be run on the input data.

Input/Output
Defines that both INPUT and OUTPUT attributes are specified.

None Defines that the field is not to show on the user interface and that no edits are to be defined for it. Items with this setting are typically used as control data for user defined edits or as items such as the defined the submit value item.

Output
Defines that output edits will be performed on data received from the server.

Program Link
Enables an item with specified Link properties to define a link that invokes a referenced program when the generated HTML link is selected by the program user.

Submit
Defines an item to contain a value, or set of values if the item has an occurs value greater than one, that can be received into the submit value item when the program user submits a form back to the server.

Submit Bypass
Defines an item to contain a value, or set of values if the item has an occurs value greater than one, that can be received into the submit value item when the program user submits a form back to the server. All input edits are bypassed when the program user submits the form.

Definition considerations for Data item UI type

The following properties are available:

- Occurrences item
- Selected index item
- Help text
- UI label

Occurrences item
The data item that defines the number of rows to display in the generated HTML page. The specified occurrences item must be defined in the UI record as follows:

Occurs
1

Data Type
Bin, Num, Numc, Pacf or Pack

Decimals
0

The data item specified as the occurrences item must not be the item you are currently defining, the selected index item of the item you are currently defining or the record's submit value item.

Selected index item
The data item that receives the index or indices of the element(s) selected by the program user. The specified selected index item must be defined in the UI record as follows:

Data Type
Bin, Num, Numc, Pacf or Pack

Decimals
0

You can specify an array item as the selected index item. If the specified selected index item is an array item, the generated UI part is a multiple select list. The occurs value of the array item must match the occurs value of the data item you are defining. If the specified selected index item is not an array item, the generated UI part is a single select list.

Help text
Help text defined for the item. Help text can be shared among all records that use a shared data item.

UI label
The label defined for the item. If the item has an occurs value greater than one and the item type is Submit, Submit Bypass or Program Link, labels can be defined for each occurrence.

Target environments for Data item UI type

See the individual data item UI types for compatibility considerations.

Data item UI type - Form

UI Type - Form is similar to Data item UI type - Program Link except that it allows items that are substructured under an item with this UI Type to be part of an HTML "Form". With UI Type - Form you can substructure items of UI Type=Input, Input/Output, Submit, etc. to create multiple Forms that invoke independent programs, passing the data of the given form.

Data item UI type - Form

Uses

Use UIType - Form if data to be passed into the referenced program when this Form is submitted is meant to be updated by the program user. The substructured items of UI Type - Input, Input/Output will be input fields that can be updated by the program user. This data will be passed to the referenced program when this Form is submitted.

Note that not all data that is substructured will be passed. Only data of UI Types Input, Input/Output, Submit, Submit Bypass, and Hidden will be passed because these types turn into forms of the HTML INPUT tag. Other UITypes such as Output can be substructured to control the appearance of the form, but this data will not be passed. To pass data of fields with UITypes of Output, None (any types that do not become HTML INPUT fields), use the Link Parameter definition as outlined in 234.

The UI record as a whole is treated as a default Form with the referenced program implicitly being the one that CONVERSEd it. The same rules as outlined previously for passing data apply to this default Form, that is, only those fields that become HTML INPUT fields will actually be passed back from the browser.

The main difference between using fields with UIType - Form and the default Form of the entire UI record is that the default Form is sent to the browser as the result of a CONVERSE and the entire state of that UI record has been saved at the server. When you invoke the reference program of an item with UIType - Form, this program is started new each time, so the First UI record of this program will only have the state of the data that is passed to it.

Definition considerations for Data item UI type - Form

The following Program Link properties are available for Data item UI type - Form:

- First UI record
- Link parameters
- Open as new window
- Program

First UI record

The name of a UI record that is defined in the specified program. If data is passed when the program is invoked, the First UI record specified contains the definition of the data items that receive data. Specifying the First UI record is optional but using a Form to invoke a program without passing data is not efficient. If the you want to invoke a program without passing data, use UIType - Program Link.

Link parameters

Parameters that associate a data item in the First UI record of the referenced program with data of the UI record containing the given

link. The difference between parameters defined here and those defined for UIType - Program Link is that in the generated HTML these parameters become HTML INPUT fields of TYPE=hidden instead of query parameters tacked onto a URL. You can successfully pass 400 bytes of data this way.

Name The name of the data item that receives data when the program is invoked. This data item must be defined in the specified First UI record.

Value Item
The name of the item that contains the data to be passed to the invoked program. Parameter values passed to this program when the program user submits this Form are the state of the value items at the time when the page is sent to the browser The data item specified as the value item and the program link must be defined in the same UI record. A literal can also be specified as the Value Item.

Open as new window
A boolean value used to specify whether the results returned when a user transits a link are displayed in a new window or in the current window. If you are defining a program link, specifying a new window for the linked program is optional.

Program
The name of the program to invoke. A Web Transaction program is the only valid type of program you can define for this property. This field is mandatory.

Target environments for Data item UI type - Form

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.

Data item UI type - Form

Environment	Compatibility considerations
OS/400	Not supported.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

Data item UI type - Hidden

UI Type - Hidden fields are not visible to the program user on the generated HTML page. However, the data of these fields will be passed when a form containing a hidden field is submitted.

Uses

Use UIType - Hidden if the data should not be visible to the program user but must be passed to the server program.

Target environments for Data item UI type - Hidden

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Not supported.

Environment	Compatibility considerations
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

Data item UI type - Input

UI Type - Input defines that input can be entered by the program user and that edits will be run on the input data.

Uses

Use UI Type - Input if the generated UI part initially displays no data and allows the program user to input data.

Target environments for Data item UI type - Input

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2	Valid only in UI records.

Data item UI type - Input

Environment	Compatibility considerations
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

Data item UI type - Input/Output

UI Type - Input/Output defines that both INPUT and OUTPUT attributes are specified.

Uses

Use UI Type - Input/Output if the generated UI part initially displays data and allows your user to input data.

Target environments for Data item UI type - Input/Output

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.

Environment	Compatibility considerations
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

Data item UI type - None

UI Type - None defines that both INPUT and OUTPUT attributes are specified for the data item.

Uses

Use UI Type - None if the data item generates control data. UI Type - None data items are not displayed to the program user.

Target environments for Data item UI type - Input/Output

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.

Data item UI type - None

Environment	Compatibility considerations
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

Data item UI type - Output

UI Type - Output defines that output edits will be performed on data received from the server.

Uses

Use UI Type - Output if the generated UI part displays data to the program user.

Target environments for Data item UI type - Output

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.

Environment	Compatibility considerations
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

Data item UI type - Program link

UI Type - Program Link enables an item with specified Link properties to define a link that invokes a referenced program when the generated HTML link is selected by the program user.

Uses

Use UI Type - Program Link when a hypertext link to another Web Transaction program is required.

Definition considerations for Data item UI type - Program link

The following Program Link properties are available:

- First UI record
- Link parameters
- Open as new window
- Program

First UI record

The name of a UI record that is defined in the specified Program. If data is passed when the program is invoked, the First UI record specified contains the definition of the data items that receive data. If you are defining a program link and not passing any data, specifying the First UI record is not necessary.

Link parameters

Parameter that associate a data item in the First UI record of the referenced program with the data of the UI record containing the given link. In the HTML, a program link becomes an HTML element and the link parameters become query parameters that are part of the HREF attribute of the HTML element. This means that there is an implied limit to the amount of data that can be passed and it varies depending on the browser and the web server used. If you need to pass more than 400 bytes, use the UI Type of Form.

Name The name of the data item that receives data when the program is invoked. This data item must be defined in the specified First UI record.

Value Item
The name of the item that contains the data to be passed to the invoked program. Parameter values passed to this program when the program user clicks on this link are the state of the value items when the page was sent to the browser. The data item specified as the value item and the program link must be defined in the same UI record. A literal can also be specified as the Value Item and usage rules are the same as when you use literals as operands in statements.

Open as new window
A boolean value used to specify whether the results returned when a user transits a link are displayed in a new window or in the current window. If you are defining a program link, specifying a new window for the linked program is optional. This field is optional.

Program
The name of the program to invoke. A Web transaction program is the only valid type of program you can define for this property. If you are defining a program link, specifying the program is mandatory.

Target environments for Data item UI type - Program link

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.

Environment	Compatibility considerations
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

Data item UI type - Submit

UI Type - Submit defines an item to contain a value, or set of values if the item has an occurs value greater than one, that can be received into the submit value item when the program user submits a form back to the server.

Uses

Use UI Type - Submit if the generated UI part is a push button that validates the data entered by the program user against the edits specified for each generated UI part on the HTML page.

Definition considerations for Data item UI type - Submit

EZEAID can be used as the submit value item if the values are valid EZEAID values.

An example UI Type - Submit definition and explanation of the generated output:

- A UI record data item is created and named SUBMIT_ITEM
- The item SUBMIT_ITEM is defined as UI Type - Submit
- The item SUBMIT_ITEM is further defined as CHA with an occurs value of 2
- A UI record data item is created and named SUBMIT_VALUE
- The item SUBMIT_VALUE is defined as UI Type - None
- The item SUBMIT_VALUE is further defined as the submit value item of the given UI record
- The values set into the array are 'F1' and 'F2'
- The Labels defined for the item contain 'CONTINUE' and 'CANCEL'

In this example, the default generated HTML page includes 2 submit push buttons with text containing the defined label values. The value of the actual button pressed, 'F1' or 'F2', is set into the item named SUBMIT_VALUE when the form is submitted back to the server. The program then tests the value of the item named SUBMIT_VALUE to determine if the program user wants to 'CONTINUE' or 'CANCEL'.

Submit properties

Initial Values

The item must have a value, set into the submit value item, for the buttons to be visible to the program user on the generated HTML page. In most cases, a value can be programmatically set into the submit value item. However, in the case of a First UI Record that is receiving data, no program logic is run in time to set the data. To address this situation, you can define initial values for Submit and Submit Bypass items. During run time or ITF execution, these values will be set into the item when the record or UI Bean is instantiated. If the item has an occurs value greater than one, each line is a separate value for each occurrence. Blank lines mean the occurrence at that index will have no value. If the item is arrayed and there is only one value specified, this value will be set on all occurrences.

Target environments for Data item UI type - Submit

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.

Environment	Compatibility considerations
Test Facility	Valid only in UI records.

Data item UI type - Submit bypass

UI Type - Submit Bypass defines an item to contain a value, or set of values if the item has an occurs value greater than one, that can be received into the submit value item when the program user submits a form back to the server. Submit Bypass is identical to Submit except that all input edits are bypassed when the program user submits the form.

Uses

Use UI Type - Submit Bypass if the generated UI part is a push button that does not validate the data entered by the program user against the edits specified for each generated UI part on the HTML page.

Definition considerations for Data item UI type - Submit bypass

EZEAID can be used as the submit value item if the values are valid EZEAID values.

Submit Bypass properties

Initial Values

The item must have a value, set into the submit value item, for the buttons to be visible to the program user on the generated HTML page. In most cases, a value can be programmatically set into the submit value item. However, in the case of a First UI Record that is receiving data, no program logic is run in time to set the data. To address this situation, you can define initial values for Submit and Submit Bypass items. During run time or ITF execution, these values will be set into the item when the record or UI Bean is instantiated. If the item has an occurs value greater than one, each line is a separate value for each occurrence. Blank lines means the occurrence at that index will have no value.

Target environments for Data item UI type - Submit bypass

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.

Data item UI type - Submit bypass

Environment	Compatibility considerations
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

UI record data item edits

UI record data item edits are part of data item definition and are available only for use in User Interface (UI) records and Web transaction programs.

Uses

Use UI record data item edits to specify the type of edit you want performed on the selected data item. The following UI record data item edits are available:

- Check SO/SI space
- Currency
- Currency symbol
- Edit function
- Edit type
- Edit table
- Fill character
- Fold
- Input required
- Maximum value

- Minimum input
- Minimum value
- Numeric separator
- Run edit function on web
- Sign
- Zero edit

Definition considerations for UI record data item edits

Input edits are processed in the following order or levels:

1. VAGen edits (valid numeric, range, required field, and so on).
2. Table edits
3. User Functions defined to run on the web server with the UI Record.
4. User Functions defined to run in the server program that did the CONVERSE.

VAGen edits, Table edits, and User Functions defined to run on the web server with the UI Record are run on the web server where the UI record beans are deployed. All fields are run through the input edits for each type. If all fields pass a level, the next level is processed. If all fields do not pass a level, processing stops at the level where input edits failed and the page is sent back to the browser. Fields are processed in the defined input edit order. The default input edit order is set as input items are created in the UI record, from the top of the record to the bottom of the record.

Target environments for UI record data item edits

See the individual UI record data item edits for compatibility considerations.

UI record data item edits - Check SO/SI space

UI record data item edits - Check SO/SI space determines whether mixed data (SBCS or DBCS) entered in the generated UI part can be converted to the mainframe SO/SI format with a valid length for the generated UI part.

Definition considerations for UI record data item edits - Check SO/SI space

Mixed fields require fewer bytes of storage on OS/2 and Windows systems because the ASCII DBCS format does not use SO/SI escape characters for delimiting DBCS data.

Check SO/SI space is only available for mixed data items.

Target environments for UI record data item edits - Check SO/SI space

Environment	Compatibility considerations
VM CMS	Not supported.

UI record data item edits - Check SO/SI space

Environment	Compatibility considerations
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

UI record data item edits - Currency

UI record data item edits - Currency displays a currency symbol or accepts a currency symbol for numeric data in the generated UI part when the program user submits the generated HTML page.

Definition considerations for UI record data item edits - Currency

One currency symbol is accepted preceding or following the numeric data entered by the program user.

Field length is calculated automatically.

Target environments for UI record data item edits - Currency

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

UI record data item edits - Currency symbol

UI record data item edits - Currency symbol defines a one to three character symbol for the data item.

Definition considerations for UI record data item edits - Currency symbol

The default system currency symbol is used when a currency symbol is not defined and UI record data item edits - Currency is defined. The default system currency symbol can be changed by your system administrator using the customization procedures for language-dependent options.

UI record data item edits - Currency symbol

Target environments for UI record data item edits - Currency symbol

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

UI record data item edits - Edit function

UI record data item edits - Edit Function defines an edit function for validating data entered by the program user in the generated user interface part.

Uses

Use edit functions to check user input entered into the generated user interface.

Definition considerations for UI record data item edits - Edit function

The edit function cannot be defined with parameters. The function can invoke other functions defined with parameters. EZEC10 and EZEC11 are also valid edit functions.

Target environments for UI record data item edits - Edit function

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Valid only in UI records.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

UI record data item edits - Edit type

UI record data item edits - Edit type defines the data item as one of the following:

- Date
- Time

- Boolean
- None

Date Date edit type specifies that the internal data of the item should be treated as a date. The purpose of this is to allow the Java Server Page developer to format it appropriately. In order for this to work the internal date format defined for the server must match the date format of the hptDateMask initialization parameter of the Gateway Servlet on the Web Server. For additional information on data item date formats, see "EZEDTELC" on page 542. If the date format of the server is sufficient i.e. it need not be specially formatted on the Java Server Page, then do not specify this edit. For additional information on date formatting for the Gateway Servlet, see the *VisualAge Generator User's Guide*.

Time Time defines the data item as a time.

Boolean
Boolean defines the data item as a boolean edit.

None None defines the data item as neither a date, time or boolean.

Definition considerations for UI record data item edits - Edit type

Valid numeric item values for the Boolean Edit Type are 1 (true) and 0 (false). Valid character item values are Y (true) and N (false).

The data item specified as Date, Time, or Boolean must be defined as follows:

Data Type
Bin, Char, Num, Numc, Pacf or Pack

Decimals
0

Target environments for UI record data item edits - Edit type

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.

Environment	Compatibility considerations
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Valid only in UI records.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

UI record data item edits - Edit table

UI record data item edits - Edit table defines an edit table for validating data entered by the program user in the generated user interface part.

Target environments for UI record data item edits - Edit table

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Valid only in UI records.
OS/2	Valid only in UI records.

UI record data item edits - Edit table

Environment	Compatibility considerations
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

UI record data item edits - Fill character

UI record data item edits - Fill character defines a character to fill unused positions in the generated UI part.

Definition considerations for UI record data item edits - Fill character

A fill character can be an alphanumeric character or a blank.

The Fill Character field is not available for DBCS or Unicode items because blank is the only valid definition.

Null is not a valid character.

Target environments for UI record data item edits - Fill character

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.

Environment	Compatibility considerations
OS/400	Valid only in UI records.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

UI record data item edits - Fold

UI record data item edits - Fold specifies the folding of alphabetic characters in the generated UI part to uppercase characters when the program user submits the generated HTML page.

Definition considerations for UI record data item edits - Fold

Fold is not available when the data type is:

- Bin
- DBCS
- Num
- Numc
- Pacf
- Pack
- Unicode

Folding does not occur for DBCS data in mixed fields.

Target environments for UI record data item edits - Fold

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.

UI record data item edits - Fold

Environment	Compatibility considerations
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Valid only in UI records.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

UI record data item edits - Input required

UI record data item edits - Input required defines that the program user must enter information in the generated UI part before submitting the generated HTML page. The edit is satisfied if the field contains a value other than blanks, or zero for a numeric field.

Target environments for UI record data item edits - Input required

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.

Environment	Compatibility considerations
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Valid only in UI records.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

UI record data item edits - Maximum value

UI record data item edits - Maximum value defines the upper limit of a range of numbers that the program user can enter in the generated UI part.

Definition considerations for UI record data item edits - Maximum value

If you specify UI record data item edits - Maximum Value, you must also specify UI record data item edits - Minimum Value.

Target environments for UI record data item edits - Maximum value

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.

UI record data item edits - Maximum value

Environment	Compatibility considerations
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Valid only in UI records.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

UI record data item edits - Minimum input

UI record data item edits - Minimum input defines the minimum number of characters that the program user is required to enter in the generated user interface part. If the program user enters any data in the generated user interface part, the minimum input definition applies.

Target environments for UI record data item edits - Minimum input

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Valid only in UI records.

Environment	Compatibility considerations
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

UI record data item edits - Minimum value

UI record data item edits - Minimum value defines the lower limit of a range of numbers that the program user can enter in the generated UI part.

Definition considerations for UI record data item edits - Minimum value

If you specify UI record data item edits - Minimum Value, you must also specify UI record data item edits - Maximum Value.

Target environments for UI record data item edits - Minimum value

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Valid only in UI records.
OS/2	Valid only in UI records.

UI record data item edits - Minimum value

Environment	Compatibility considerations
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

UI record data item edits - Numeric Separator

UI record data item edits - Numeric Separator accepts on input or displays a numeric separator for numeric data.

Definition considerations for UI record data item edits - Numeric Separator

Field length is calculated automatically.

If the number of significant digits is fewer than 4, Numeric Separator is not valid.

For VisualAge Generator Developer, the default numeric separator is determined by your development environment's system setting.

Target environments for UI record data item edits - Numeric Separator

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.

Environment	Compatibility considerations
CICS for OS/2	Not supported.
OS/400	Valid only in UI records.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

UI record data item edits - Run edit function on web

UI record data item edits - Run edit function on web defines that the edit function is to run on the web server where the UI record run time bean is deployed.

Definition considerations for UI record data item edits - Run edit function on web

When Run edit function on web is defined, the function's data usage is strictly limited to the data of the UI record where the function is defined. When Run edit function on web is not defined, the edit function runs on the server and can access any of the data available to the program.

Target environments for UI record data item edits - Run edit function on web

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.

UI record data item edits - Run edit function on web

Environment	Compatibility considerations
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Valid only in UI records.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

UI record data item edits - Sign

UI record data item edits - Sign defines how the program user enters a sign in the generated UI part:

Leading
Accepts on input or displays a plus (+) or a minus (-) sign to the left of numeric data.

Trailing
Accepts on input or displays a plus (+) or a minus (-) sign to the right of numeric data.

None Prevents your user from entering a sign with the numeric data.

Definition considerations for UI record data item edits - Sign

Field length is calculated automatically.

Target environments for UI record data item edits - Sign

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.

Environment	Compatibility considerations
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Valid only in UI records.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

UI record data item edits - Zero edit

UI record data item edits - Zero edit displays the value zero in the generated UI part.

Definition considerations for UI record data item edits - Zero edit

If UI record data item edits - Zero Edit is not defined, the value zero is not displayed in generated UI part.

Target environments for UI record data item edits - Zero edit

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Valid only in UI records.
MVS/TSO	Not supported.

Environment	Compatibility considerations
MVS batch	Not supported.
IMS/VS	Valid only in UI records.
IMS BMP	Not supported.
CICS for VSE/ESA	Valid only in UI records.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Valid only in UI records.
OS/2	Valid only in UI records.
AIX	Valid only in UI records.
HP-UX	Valid only in UI records.
CICS for AIX	Valid only in UI records.
Windows NT	Valid only in UI records.
CICS for Windows NT	Valid only in UI records.
Solaris	Valid only in UI records.
CICS for Solaris	Valid only in UI records.
Test Facility	Valid only in UI records.

Chapter 7. Program specification block

A program specification block (PSB) is a formal DL/I description of the hierarchical database structures a program can access. VisualAge Generator uses the PSB definition to build and validate DL/I calls for I/O functions that access records in DL/I databases. The PCBs are listed in the VisualAge Generator PSB in the same order that they appear in the actual DL/I or IMS PSB definition to be used with the program. The PSB structure also identifies the PCBs used for terminal, printer, and message queue support in the IMS/VS and IMS BMP environments.

VisualAge Generator supports the definition of a part that contains a subset of the information in the DL/I PSB. The PSB definition describes the hierarchical relationship between types of segments.

A PSB is made up of program communication blocks (PCBs). You define the PSB by defining its PCBs.

Program specification block elements

Table 14. Program specification block elements

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
Program specification block			c	c	c	c	c	c	c													x
Program communication block			c	c	c	c	c	c	c													x
Note: *Includes Windows 95 and Windows 98, Windows NT, and Windows 2000.																						

Table 14. Program specification block elements (continued)

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
Note: In this table, the following characters are used to indicate the level of support: **x** Supported **c** Supported with compatibility considerations **blank** Not supported																						

Program communication block (PCB)

A PCB is an entry in a PSB that describes the segment within one hierarchical database, IMS message queues, or GSAM databases.

Uses

You can specify the following information when defining PCBs:

PCB number
The number of the PCB you are defining. The PCB number is calculated by VisualAge Generator, based on the placement of the PCB within the PSB.

The number identifies which PCB in the PSB is to be used when the database name appears in more than one PCB in the PSB definition.

Type There are three types of PCBs: database (DB); generalized sequential access method (GSAM); and teleprocessing (TP).

The three types of PCBs are described under "Definition considerations for PCBs" on page 267.

Database
A 1- to 8-character name of the database used with the PSB. The database name is required for DB and GSAM PCBs, but cannot be specified with TP PCBs.

When you specify a new database, you signal the start of a new PCB in the PSB.

You can also specify ELAMSG or ELAWORK as the name of the database. ELAMSG and ELAWORK represent message and work databases used with Server for MVS, VSE, and VM to run the

program. If you specify ELAMSG or ELAWORK as the name of the database, do not specify a segment or index key.

Segment
A 1- to 8-character name for the segment in the database. The segment name must be the same as the name in the DL/I PSB.

Note: Before you can access the segment in a program, you must define the segment as a record with the DL/I segment organization.

Parent A 1- to 8-character name of the segment that is the parent of this segment in the database. The parent/segment relationship (hierarchy) must be the same as in the DL/I PSB.

If the segment is the root segment in a PCB structure, the parent name is blank.

Index key
A 1- to 8-character name for the secondary index key field.

If you want the program to access the database through a secondary index (the PCB in the DL/I PSB has a PROCSEQ keyword specified), you must name the secondary index key.

The index key must be the name of a data item you have defined in a segment.

The data item name must be the same as the name specified for the secondary index field in the DL/I database description (NAME keyword in the XDFLD statement). The data item length must be the same as the length of the field defined in the XDFLD statement.

Definition considerations for PCBs

You can pass individual PCBs on a call. This enables you to define a program with a PSB and call the program from other programs that have different PSB structures. You use the special function word EZEDLPCB, subscripted with the PCB number to be passed, on the CALL statement.

The following describes the three types of PCBs:

Database (DB) PCB
Each DB PCB describes one hierarchical data structure that a program can use. The data structure might correspond directly to the structure of a physical or logical DL/I database or might invert the database structure through access by a secondary index.

If the database is accessed using a secondary index, the first line must contain the PCB type (DB), the database name, the name of the root segment, and the name of the index field.

One line is specified for each SENSEG segment defined for the database PCB in the DL/I or IMS Program Specification Block. Each SENSEG line specifies a segment name and parent name in the same order that they appear in the PSB. If the segment is the root segment (no parent) in a PCB structure, the database name is specified, and the parent name is left blank.

For more information on DB PCBs, refer to "Developing DL/I Programs" in *VisualAge Generator Design Guide*

Generalized Sequential Access Method (GSAM) PCB

Each GSAM PCB represents a generalized sequential access method (GSAM) database in an IMS Program Specification Block.

The database name is the only field that can be specified for a GSAM PCB. GSAM PCBs appear last in the PSB definition.

Teleprocessing (TP) PCB

Each TP PCB represents an alternate PCB in an IMS Program Specification Block. The alternate PCB represents a terminal, printer, or message queue in the IMS environment.

There is one line in the list for each teleprocessing PCB. TP PCBs appear first in the PSB definition.

A TP PCB must not be specified for PCB zero, which is the main I/O PCB. This PCB is not specified in the IMS PSB definition.

TP PCBs are not used in non-IMS environments but can be included if the program using the PSB is to be generated for both IMS and non-IMS environments.

Target environments for PCBs

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	The I/O, teleprocessing (TP), and GSAM PCBs cannot be referenced in the CICS environment with the EZEDLPCB special function word, or the CSPTDLI service routine. When database (DB) PCBs are accessed using the EZEDLPCB special function word or the CSPTDLI service routine, the PCB number should be specified using the PCB number from the PSB definition.

Environment	Compatibility considerations
MVS/TSO	The IMS PSB definition must be generated with the CMPAT=YES option specified on the PSBGEN macro. The TP PCBs are ignored. At least two PCBs are needed in the PSB so the COBOL program can determine whether it is being started by the IMS control region or through an XCTL from a non-VisualAge Generator program passing working storage and the EZEDLPSB parameter.
MVS batch	Same as MVS/TSO.
IMS/VS	The IMS PSB generated for use with a main transaction program must have the same name as the COBOL program load module for the program. The default name for the load module is the program name. The definition for the IMS PSB must match the definition of the PSB part PCB for PCB, except for the PSB name and the database names. The first TP PCB must be a modifiable alternate PCB, to be used for switching transactions. The second TP PCB must be a modifiable express alternate PCB, to be used for diagnostic information. These two TP PCBs are required. Additional modifiable or not modifiable alternate or express alternate PCBs can follow. Database PCBs must be included in the PSB for the VisualAge Generator Server for MVS, VSE, and VM work database (ELAWORK) if a DL/I implementation of this database is used. The work database is indicated in the PSB by database name only and does not require a line for each segment.
IMS BMP	The IMS PSB definition must be generated with the CMPAT=YES option specified on the PSBGEN macro. The TP PCBs are used to access the message queues as serial files. Programs that read input from the I/O PCB are transaction-oriented BMPs. Programs that do not read input from the I/O PCB are batch-oriented BMPs. The requirements defined for the two TP PCBs for the IMS/VS environment also apply.
CICS for VSE/ESA	Same as CICS for MVS/ESA.

Environment	Compatibility considerations
VSE batch	DL/I DOS/VS does not support definition of PSBs with teleprocessing (TP) or GSAM PCBs. If a PSB with teleprocessing PCBs will be used in the same program on both MVS and VSE systems, the DL/I DOS/VS PSB should omit the TP and GSAM PCBs. When using a language element which requires specifying a PCB by number (such as EZEDLPCB or CSPTDLI), always use the PCB number for the PCB as it is defined in the PSB part. The VisualAge Generator Developer adjusts the number you specify to account for the TP and GSAM PCBs not being included in the DL/I DOS/VS PSB.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
CICS for AIX	Not supported.
Windows NT	Not supported.
CICS for Windows NT	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
Test Facility	None.

Chapter 8. Maps

A map is a format specification for a character-based user interface on a terminal or printer. Map specification language elements specify nonprocedural processing information related to a map.

Map specification enables you to define options such as the size and position of a map. The map size is the number of lines and columns of a map. The map position is the position where the map starts on a device. Other options in map specification include items such as whether to fold input to uppercase, the name of the help map, and which key is the help key.

Map elements

Table 15. Map elements

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
Bypass edit keys	x		x	c		c		x		x	x			x	x	x	x	x	x	x	x	x
Device selection	x	x	x	x	x	c	c	x	x	c	c			c	c	c	c	c	c	c	c	x
Floating area	x	x	x	x	x	c	c	x	x	x	x			c	c	c	c	c	c	c	c	x
Floating map	x	x	x	x	x	c	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Initial cursor field	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	x
Help key	x		x	c		c		x		x	x			x	x	x	x	x	x	x	x	x
Help map name	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	x
Map group	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Map name	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Map position	x	x	x	x	x	x	x	x	x	x	c			x	x	x	x	x	x	x	x	x

Table 15. Map elements (continued)

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
Map size	x	x	x	x	x	c	x	x	x	x	x			x	x	x	x	x	x	x	x	x
SO/SI take position	c	c	c	c	c	c	c	c	c	c	c			c	c	c	c	c	c	c	c	c
Variable field folding	x		c	x		c		c		c	x			c	c	c	c	c	c	c	c	x

Note: *Includes Windows 95 and Windows 98, Windows NT, and Windows 2000.

Legend: In this table, the following characters are used to indicate the level of support:
x Supported
c Supported with compatibility considerations
blank Not supported

Bypass edit keys

Bypass edit keys enable the program user to to bypass map edits and map edit routines.

Uses

You can specify up to five function keys as bypass edit keys. PA keys are treated as bypass edit keys in a generated program.

When the program user presses a bypass edit key, data is not passed to the program and the program continues processing at the statement following the terminal I/O function (either the first map or an I/O option).

The data on the map is not saved when the program user presses a bypass edit key.

During program specification you can specify bypass edit keys to be used as defaults for all the maps used by a program. The bypass edit keys you define for the map override the default specification. For example, if you specify three keys to be bypass edit keys, but you only specify one bypass edit key on the map you define, only that key can be used for that map.

Note: You cannot have a function key be both a bypass edit key and the help key.

Target environments for Bypass edit keys

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	PF6 is reserved for a panel recovery function in this environment. Pressing PF6 is treated as pressing the Clear key. The PF key value is not passed back to the program. Avoid using PF6 in this environment.
MVS batch	Not supported.
IMS/VS	IMS reserves the PA keys so they cannot be the default bypass edit keys. A specific PF key must be defined if the program user is allowed to bypass edits. If your installation uses PF12 for the IMS local copy function, PF12 cannot be used as a bypass edit key.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Device selection

Device selection enables you to specify the devices on which the map can be displayed or printed.

You must always have at least one device supported at all times.

Definition considerations for Device selection

Many of the device types are supported only for compatibility with previous products. If you are defining a new map, select Printer (SBCS) or 5550P (DBCS) for a print map, 5550D for a DBCS terminal map, or an ANY-xx device with the correct screen size for a single-byte terminal map.

A map group that does not contain any DBCS maps cannot be used in the same job step with a map group that contains DBCS maps. Make sure that at least one map in the map group specifies a DBCS device type if you mix DBCS and non-DBCS maps from different map groups.

Target environments for Device selection

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	For non-DBCS terminals, only 3270-type terminals are supported. If non-3270 display devices (3643) are specified, generation builds MFS definitions for 3270 devices with a compatible screen size. The minimum screen size supported is 24 x 80. If maps are defined with a smaller screen size, generation builds MFS definitions for 3270 devices with a 24 x 80 screen. The MFSDEV, MFSFEAT, and MFSEATTR generation options should be set up to specify the device characteristics for different device types at your installation.
IMS BMP	If you specify the MSP(MFS) or MSP(ALL) generation option, the considerations are the same as IMS/VS. Otherwise there are no compatibility considerations.
CICS for VSE/ESA	None.
VSE batch	None.

Environment	Compatibility Considerations
CICS for OS/2	CICS OS/2 3270 emulation can display maps on the workstation screen, in a window, or on an ASCII terminal attached to the workstation as specified in the CICS OS/2 Terminal Control Table (TCT). The screen dimensions are specified in the TCT entry. When specifying a display device for a map in map definition, specify a device with the same screen dimensions specified in the TCT entry.

OS/400	The following device types are supported:
	`ANY-2D 24 x 80 panel`
	`ANY-5D 27 x 132 panel`
	`PRINTER printer`
	`5550D DBCS terminal`
	`5550P DBCS printer`

OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	There is no concept of a Device when generating C++ programs. The C++ runtime will use the current size of the OS/2 session, Windows NT session, or AIX window when running a program. If the maps are defined with variable fields that wrap on multiple lines, the program user should define the OS/2 session, Windows NT session, or AIX window with the same number of columns as the map. Otherwise, the wrapping variable fields can have unpredictable results.
AIX	Same as OS/2 (C++).
HP-UX	Same as OS/2 (C++).
CICS for AIX	CICS for AIX 3270 emulation can display maps on the workstation screen, in a window, or on an ASCII terminal attached to the RS/6000 as specified in the CICS for AIX Terminal Control Table (TCT). The screen dimensions are specified in the TCT entry. When specifying a display device for a map in map definition, specify a device with the same screen dimensions specified in the TCT entry.
Windows NT	Same as OS/2 (C++).
CICS for Windows NT	CICS for Windows NT 3270 emulation can display maps on the workstation screen, in a window, or on an ASCII terminal attached to the Windows NT system as specified in the CICS for Windows NT Terminal Control Table (TCT). The screen dimensions are specified in the TCT entry. When specifying a display device for a map in map definition, specify a device with the same screen dimensions specified in the TCT entry.

Device selection

Environment	Compatibility Considerations
Solaris	Same as OS/2 (C++).
CICS for Solaris	CICS for Solaris 3270 emulation can display maps on the workstation screen, in a window, or on an ASCII terminal attached to the RS/6000 as specified in the CICS for Solaris Terminal Control Table (TCT). The screen dimensions are specified in the TCT entry. When specifying a display device for a map in map definition, specify a device with the same screen dimensions specified in the TCT entry.
Test Facility	None.

Floating area

Floating area is an area within a device display region reserved for displaying maps defined as floating maps.

Uses

Usually, the floating area is defined with the same depth for all terminals in a map group. No side-by-side maps are supported. Only one floating area can be specified for each device in a map group.

The floating area consists of a floating area size and a floating area position.

The following elements must be specified to define a floating area size:

Lines The number of lines in the floating area.

Columns
The number of columns in the floating area.

When a floating map appears, it is written to the next available line in the floating area defined for the map group. Once the floating area is full, the program must converse the last map so that all the displayed floating maps can be seen by the user. If there is not sufficient room in the floating area for the map, the floating area is erased and the map is positioned in the first line of the floating area.

The following elements must be specified to define a floating area position:

Starting line
The starting line of the floating area.

Starting column
The starting column of the floating area.

Note: If you specify a value for any one of the elements above, you must specify a value for all the elements. If these values are left blank, the entire device is considered as the floating area.

Target environments for Floating area

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	A floating area is valid only for printer maps. A floating area is not valid for maps defined for display devices.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	Only one floating area is used for display maps and only one floating area is used for printer maps. If different floating areas are defined for different display or printer types, the definition that is used depends on the order in which maps were defined, and is unpredictable. To avoid confusion, specify the same print device for all printer maps and the same display device for all display maps, or specify the same floating area for all print devices and the same floating area for all display devices.
AIX	Same as OS/2 (C++).
HP-UX	Same as OS/2 (C++).
CICS for AIX	Same as OS/2 (C++).
Windows NT	Same as OS/2 (C++).
CICS for Windows NT	Same as OS/2 (C++).
Solaris	Same as OS/2 (C++).
CICS for Solaris	Same as OS/2 (C++).
Test Facility	None.

Floating map

When you specify a map as a floating map, you can specify only the size of the map, not the position. The starting line of the map is "Next" by definition, meaning the map will occupy the next available line in the floating area. The starting column of the map is "Same" by definition, meaning the map always displays in the same column, which is defined by the position of the floating area.

Definition considerations for Floating map

A floating map is displayed starting at the next available line in the floating area. If the map will not fit in the remaining available lines, the floating area is erased and the map is displayed at the top of the floating area.

To ensure that the user sees all floating maps, code your program to issue a CONVERSE instead of a DISPLAY for the last map that will fit in a floating area. Maps written with the DISPLAY option do not show up on the screen until the next CONVERSE and will be lost if a subsequent DISPLAY or CONVERSE causes the floating area to be erased.

When a floating print map is displayed following a fixed print map, a page eject occurs and the floating map displays in the first line of the floating area. When a fixed print map is displayed following a floating map, a page eject is issued before the fixed map is displayed.

Target environments for Floating map

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	A floating map is valid only for printer maps. A floating map is not valid for maps defined for display devices.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.

Environment	Compatibility Considerations
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Initial cursor field

The initial cursor field is the field on a map where the cursor appears when the map is first displayed.

Uses

The default for the initial cursor field is the first named and unprotected variable field on the map.

Definition considerations for Initial cursor field

To define a variable field as the initial cursor field, select the initial cursor field attribute for the field where you want the cursor to appear. Alternatively, from the **Define** menu in the Map editor, select **Field Edit Order**, then **Show Tags** and place the initial cursor graphical tag on the field you want to initially set the cursor on. The initial cursor graphical tag is the yellow tag.

At runtime, the `SET item CURSOR` statement overrides the initial cursor field previously specified.

Target environments for Initial cursor field

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.

Initial cursor field

Environment	Compatibility Considerations
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Help key

Help key specifies a function key that displays the help map for a map.

Uses

If you do not specify a value for a help key within map definition, the value specified in program definition is used as the default.

If you specify a help key, you must also specify a help map name.

Target environments for Help key

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.

Environment	Compatibility Considerations
CICS for MVS/ESA	None.
MVS/TSO	PF6 is reserved for a panel recovery function in this environment. If you press PF6, it is treated as pressing the Clear key. The PF key value is not passed back to the program. Avoid using PF6. in this environment.
MVS batch	Not supported.
IMS/VS	If your installation uses PF12 for the IMS local copy function, PF12 cannot be used as a help key.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Help map name

Help map name specifies the help map that displays when the user presses the help key while conversing the map.

Uses

The map must be in the help map group specified for the program. If a help map group is not specified, the map must reside in the program map group.

Definition considerations for Help map name

A help map is a map defined with the following restrictions:

- A help map cannot have variable fields.
- A help map cannot be a floating map.
- A help map must be defined for display, not printing.

The screen is always erased prior to the display of a help map.

Target environments for Help map name

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Map group

A map group has two different meanings in VisualAge Generator. In one context, a map group is a logical grouping of maps that are used by a particular program as I/O objects or help maps. In the other context, a map group is a part that contains the definition of the floating areas to be used for the various devices supported by the maps in the map group. In both cases, the name of the map group is the first part of the two part name that is specified when saving a map part.

Uses

Maps in a map group can be shared between VisualAge Generator programs.

You specify a map group name in the program to indicate the set of maps that can be used by the program, either as I/O objects or as the FIRSTMAP. You can optionally specify a help map group name to indicate the set of help maps that can be used by the maps displayed by the program.

To define floating areas, open the Map Group Editor and define the floating areas for each supported device. Any maps in this map group with the device specified can use the floating areas.

The generated map group is the combination of the executable form of the maps in the map group, as well as the floating area information required to properly display floating maps from a program. The map group can be generated with or separate from the programs in which it is used.

Definition considerations for Map group

Each map within a map group must have a unique name. All maps used in a program must be in the same map group, except for help maps, which can be in a separate map group.

The map group name and the map name are separated by a blank. The format for a map group name is as follows:

Map group name
Naming conventions for map groups:

Maximum length
6

First character
Alphabetic (A-Z)

Other characters
Alphanumeric (A-Z, 0-9)

DBCS name
No

- The map group name cannot have the same name as the map
- The map group name cannot have the same name as another program in the MVS library or the same CICS system

The following part name conventions apply to all part types:
- part names cannot begin with the EZE prefix.
- part names cannot contain embedded blanks.
- part names cannot be COBOL reserved words (in COBOL environments)

Target environments for Map group

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	All maps in a map group are generated in a single MFS message input description (MID) or message output description (MOD) per device. If you have a large map group, it can exceed the 32K size limit for MFS control blocks. For more information on estimating the size, refer to the *Design Guide* document.
IMS BMP	If you specify the MSP(MFS) or the MSP(ALL) generation option, the IMS/VS compatibility considerations apply. Otherwise, there are no compatibility considerations.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.

Environment	Compatibility Considerations
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Map name

Map name specifies the unique name of a map that is used to define the layout and characteristics of information to be presented on a screen or printed report.

Definition considerations for Map name

The format for map name is as follows:

Map name
Naming conventions for maps:

Maximum length
8

First character
Alphabetic (A-Z) or one of the valid national characters for your workstation

Other characters
Alphanumeric (A-Z, 0-9), or one of the valid national characters for your workstation

DBCS name
No

The following part name conventions apply to all part types:
- part names cannot begin with the EZE prefix.
- part names cannot contain embedded blanks.
- part names cannot be COBOL reserved words (in COBOL environments)

Target environments for Map name

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.

Map name

Environment	Compatibility Considerations
IMS/VS	All maps in a map group are generated in a single MFS message input description (MID) or message output description (MOD) per device. If you have a large map group, it can exceed the 32K size limit for MFS control blocks. For more information on estimating the size, refer to the *Design Guide* document.
IMS BMP	If you specify the MSP(MFS) or the MSP(ALL) generation option, the IMS/VS compatibility considerations apply. Otherwise, there are no compatibility considerations.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Map position

Map position specifies the starting position on a device using a line and column coordinate.

Uses

The default is line 1 column 1 (upper left corner).

You can specify the following:

Starting line
The row on the device where the map begins.

Starting column
The column on the device where the map begins.

If you specify a Floating map, you cannot specify a starting line or starting column. The starting line is set to Next and the starting column is set to Same.

Definition considerations for Map position

If the position and size of the partial maps (maps smaller than the device size) permit each map to display in a different set of rows on the screen, then partial maps can share the same device.

If one fixed map overlaps another perfectly, the screen is not erased. To overlap perfectly, both maps must be the same size and start in the same position.

If one fixed map overlaps another imperfectly, the screen is erased. Even if a map is positioned in the same rows (side-by-side, with no overlap), the screen is erased.

Target environments for Map position

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.

Map position

Environment	Compatibility Considerations
OS/400	The following compatibility considerations apply to maps displayed on devices, including DBCS devices, in the 5250 family: • Row 1, column 1, must either be blank or contain a field attribute byte. • The number of variable fields allowed varies with the control unit to which the display device is attached. If the number is less than or equal to 256, the map can be displayed on any 5250 control unit. The following compatibility considerations apply to maps containing double-byte fields: • An AS/400 DBCS screen does not display double-byte characters that start in column 80. Instead, a single-byte X is displayed in column 80 and in column 1 of the following line. To avoid this situation, do not define double-byte fields that span lines. • When field outlining is specified for a field on a map, no data other than blanks can be displayed in the first three bytes on a map (row 1, columns 1 through 3). In addition, row 1, column 4 can only be a blank or an attribute byte.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Map size

Map size specifies the number of lines and columns for a map.

Uses

You can specify the following:

Lines The depth of the map.

Columns
The width of the map.

The default size is the maximum size that fits on all selected devices. You can change the size to be less than the default size, defining a partial map.

Target environments for Map size

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	Because the screen is cleared after each CONVERSE function is processed, multiple partial maps are not supported for display maps.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

SO/SI take position

SO/SI take position enables you to specify that shift-out (SO) and shift-in (SI) characters take a position when a DBCS printer map is printed.

Uses

This is the default setting for a new map.

To line up the information on your maps like they are in the map definition, do the following:

- Specify SO/SI take position if you are using DBCS printers that strip the SO/SI characters from the print line.
- Do not specify SO/SI take position if you are using DBCS printers that automatically print SO/SI characters as blanks.

Target environments for SO/SI take position

Environment	Compatibility Considerations
VM CMS	The EBCDIC format for DBCS data is used, in which each DBCS string is preceded by an SO character and followed by an SI character. When directing printer output containing DBCS or mixed data to a system printer on these environments, you must specify to the system print utility that SO and SI will not take a position. If you want the SO/SI position to be represented by blanks in the printed output, specify SO/SI take position. The program sends a blank with each SO/SI in a mixed field in the output file. If you do not want the SO/SI position represented by blanks, do not specify SO/SI take position.
VM batch	Same as VM CMS.
CICS for MVS/ESA	Same as VM CMS.
MVS/TSO	Same as VM CMS.
MVS batch	Same as VM CMS.
IMS/VS	Same as VM CMS.
IMS BMP	Same as VM CMS.
CICS for VSE/ESA	Same as VM CMS.
VSE batch	Same as VM CMS.

Environment	Compatibility Considerations
CICS for OS/2	OS/2 supports the ASCII format for DBCS data which does not use SO/SI delimiters. SO/SI take position is ignored. SO/SI characters in mixed constant fields on the map are always replaced with blanks, so that the constant fields on the map appear just as you defined them in map definition. No blanks are inserted around DBCS substrings in mixed variable fields on the map.
OS/400	Same as VM CMS.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	OS/2 supports the ASCII format for DBCS data which does not use SO/SI delimiters. If a host program is being migrated to the workstation, specify SO/SI take position. This causes blanks to to be inserted in mixed constant fields where an SO or SI character would have appeared on the host. Otherwise, the constant fields appear in native format. No blanks are inserted around DBCS substrings in mixed variable fields on the map.
AIX	Same as OS/2 (C++).
HP-UX	Same as OS/2 (C++).
CICS for AIX	Same as OS/2 (C++).
Windows NT	Same as OS/2 (C++).
CICS for Windows NT	Same as OS/2 (C++).
Solaris	Same as OS/2 (C++).
CICS for Solaris	Same as OS/2 (C++).
Test Facility	Same as OS/2 (C++).

Variable field folding

Variable field folding changes all non-numeric, SBCS data entered in all variable fields to uppercase.

Uses

This is the default setting for a new map.

Do not use Variable field folding for only specific variable fields. To have data folded to uppercase for specific fields, use the Fold variable field edit.

Target environments for Variable field folding

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	Do not specify UCTRAN in the CICS terminal definition if you want lower case data to be passed to the program.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	For folding to be effective, EDIT=ULC must be specified on the TRANSACT macro in the IMS GEN. If EDIT=UC, input data will be folded by IMS. If EDIT names a user-supplied transaction input edit routine, the data is edited by that routine and then edited based on any folding requirements specified for the map.
IMS BMP	Not supported.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	Not supported.
CICS for OS/2	For folding to be effective, do not specify UCTRAN. Folding is performed based on the current code page in effect for OS/2.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	Folding is performed based on the current code page.
AIX	Same as OS/2 (C++).
HP-UX	Same as OS/2 (C++).
CICS for AIX	Same as OS/2 (C++).
Windows NT	Same as OS/2 (C++).
CICS for Windows NT	Same as OS/2 (C++).
Solaris	Same as OS/2 (C++).
CICS for Solaris	Same as OS/2 (C++).
Test Facility	None.

Chapter 9. Map fields

Map field specification, using nonprocedural elements, enables you to define the fields on a map and the various edits that can be associated with those fields. Map fields can either be constant or variable. You can define editing characteristics for each field. Field attributes, such as BRIGHT or RED, can be specified for both constant and variable fields.

Map field elements

Table 16. Map field elements

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
Constant field	x	x	x	x	x	c	c	x	x	x	c			x	x	x	x	x	x	x	x	x
Constant field - DBCS	x	x	x	x	x	c	c	x	x	c	c			c	c	c	c	c	c	c	c	x
Constant field - Mixed	x	x	x	x	x	c	c	x	x	c	c			c	c	c	c	c	c	c	c	x
Field attribute - Color	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	x
Field attribute - Highlight	x		x	x		x		x		c	c			c	c	c	c	c	c	c	c	x
Field attribute - Initial cursor field	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	x
Field attribute - Input required	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	x

Table 16. Map field elements (continued)

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
Field attribute - Intensity	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	x
Field attribute - Light pen detect	x		x	x		x		x		c	x			c	c	c	c	c	c	c	c	x
Field attribute - Modified data tag	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	x
Field attribute - Numeric	x		x	x		x		x		x	x											x
Field attribute - Outlining	x	x	x	x	x	x	x	x	x	c	c			c	c	c	c	c	c	c	c	x
Field attribute - Protection	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	c
Field attribute - Require fill on input	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	x
Message field - EZEMSG	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	x
Variable field	x	x	x	x	x	c	x	x	x	x	c			x	x	x	x	x	x	x	x	x
Variable field array - Field placement	x	x	x	x	x	c	x	x	x	x	c			x	x	x	x	x	x	x	x	x

Table 16. Map field elements (continued)

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
Variable field - DBCS	x	x	x	x	x	c	x	x	x	c	c			c	c	c	c	c	c	c	c	x
Variable field - MIX	x	x	x	x	x	c	x	x	x	c	c			c	c	c	c	c	c	c	c	x
Variable field edit - Check SO/SI space	i		i	i		i		i		x	x			x	x	x	x	x	x	x	x	x
Variable field edit - Currency	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Variable field edit - Date edit mask	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Variable field edit - Decimals	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Variable field edit - Description	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Variable field edit - Edit error message number	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	x
Variable field edit - Edit routine	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	x
Variable field edit - Fill character	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x

Table 16. Map field elements (continued)

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
Variable field edit - Fold	x		c	x		c		c		c	x			c	c	c	c	c	c	c	c	x
Variable field edit - Hex edit	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	x
Variable field edit - Input required	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	x
Variable field edit - Justify	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Variable field edit - Maximum value	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	x
Variable field edit - Minimum input	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	x
Variable field edit - Minimum value	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	x
Variable field edit - Numeric separator	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Variable field edit - Sign	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x

Table 16. Map field elements (continued)

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
Variable field edit - Zero edit	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Variable field edit order	x		x	x		x		x		x	x			x	x	x	x	x	x	x	x	x
Variable field length	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x
Variable field name	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x

Note: *Includes Windows 95 and Windows 98, Windows NT, and Windows 2000.

Legend: In this table, the following characters are used to indicate the level of support:
x Supported
c Supported with compatibility considerations
i Ignored.
blank Not supported

Constant field

Constant fields on a map are fields that cannot be modified by the program.

Uses

A constant field appears enclosed in brackets. A constant field is specified by selecting a Constant object and placing it at the desired position on the map presentation area. The constant field mark specifies the starting position of a constant field, but it is not a visible character on the map. The starting field mark of the next field is considered the end mark of the previous field.

Target environments for Constant field

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.

Environment	Compatibility Considerations
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	Row 1 column 1 on the map must be blank, or contain a constant or variable field mark character. Unprotected literal constant fields are not supported. The generated MFS defines them as though they were protected. If a constant field mark character is in the last position of a map (last row, last column), it is treated as though it is in row 1 column 1. A zero-length constant field is not supported for maps. Non-blank constant fields longer than 255 bytes are not supported. Each map must contain 8 bytes in a constant field that can be used to store the IMS transaction code in the MFS definition for the map. You can designate this field by defining an 8-byte constant field on the map with the protect and dark attributes. Explicitly defining the transaction code constant on the map enables the program user to use the IMS /FORMAT command to display a formatted map to start a transaction. If you do not define an 8-byte constant, generation looks for any 9 consecutive blanks in a constant on the map and sets aside this area as a protected, dark variable field in the generated MFS map. The generated COBOL program uses this field to store the name for a subsequent IMS transaction started after a CONVERSE or XFER statement with a map. The /FORMAT command cannot be used to start a transaction for these maps because there is no default IMS transaction code. Two additional consecutive blanks in a constant field are required on each terminal map. This area is converted to a protected, nondisplay field on the map to indicate what information is stored in the work database. If you do define the transaction and flag constant fields, they can be next to each other in a single 10-byte area.
IMS BMP	If you specify the /MSP=MFS or the /MSP=ALL generation option, then IMS BMP has the same considerations as IMS/VS. Otherwise there are no compatibility considerations.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.

Environment	Compatibility Considerations
OS/400	The following compatibility considerations apply to maps displayed on devices, including DBCS devices, in the 5250 family: • Row 1, column 1, must either be blank or contain a field attribute byte. • The number of variable fields allowed varies with the control unit to which the display device is attached. If the number is less than or equal to 256, the map can be displayed on any 5250 control unit.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Constant field - DBCS

DBCS (double-byte character set) constant fields are map constants that contain double-byte characters. DBCS data is ideographic character data that requires two positions for each character.

Uses

A DBCS constant field appears enclosed in brackets. A DBCS constant field is specified by selecting a DBCS Constant part and placing it at the desired position on the map presentation area. The constant field mark specifies the starting position of a constant field, but it is not a visible character on the map. The starting field mark of the next field is considered the end mark of the previous field.

Definition considerations for Constant field - DBCS

DBCS constants can be specified only for maps defined for DBCS devices. DBCS devices are display and printer devices with DBCS capability that enable double-byte character data to be viewed or printed. Double-byte characters are required for languages such as Chinese, Japanese, and Korean.

For constant DBCS fields on a DBCS printer, wrapping fields are allowed only if the map width is equal to the printer width. If you define a wrapping field, you must start the field in an even-numbered column.

Target environments for Constant field - DBCS

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	Row 1 column 1 of the map must be blank, or contain a constant or variable field mark. Unprotected literal constant fields are not supported. The generated Message Format Service (MFS) defines them as though they were protected. If a constant field mark is in the last position of a map (last row, last column), it is treated as though it were in row 1 column 1. A zero-length constant field is not supported for maps. DBCS constant fields longer than 254 bytes are not supported.
IMS BMP	If you specify the /MSP=MFS or the /MSP=ALL generation option, then the IMS/VS considerations apply to IMS BMP. Otherwise, there are no compatibility considerations.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	A double-byte character that starts in the last column of a line is split in the middle when displayed on the workstation. To avoid this, do not define a double-byte field that spans lines on the map.
OS/400	The following compatibility considerations apply to maps containing double-byte fields: • An AS/400 DBCS screen does not display double-byte characters that start in column 80. Instead, a single-byte X is displayed in column 80 and in column 1 of the following line. To avoid this situation, do not define double-byte fields that span lines. • When field outlining is specified for a field on a map, no data other than blanks can be displayed in the first three bytes on a map (row 1, columns 1 through 3). In addition, row 1, column 4 can only be a blank or an attribute byte.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	Same as CICS for OS/2.

Environment	Compatibility Considerations
AIX	Same as CICS for OS/2.
HP-UX	Same as CICS for OS/2.
CICS for AIX	Same as CICS for OS/2.
Windows NT	Same as CICS for OS/2.
CICS for Windows NT	Same as CICS for OS/2.
Solaris	Same as CICS for OS/2.
CICS for Solaris	Same as CICS for OS/2.
Test Facility	None.

Constant field - MIX

MIX constant fields are map constants that might contain both SBCS (single-byte) and DBCS (double-byte) characters. DBCS data is ideographic character data that requires two positions for each character.

Uses

A mixed constant field appears enclosed in brackets. A mixed constant field is specified by selecting a MIXED Constant part and placing it at the desired position on the map presentation area. The constant field mark specifies the starting position of a constant field, but it is not a visible character on the map. The starting field mark of the next field is considered the end mark of the previous field.

Definition considerations for Constant field - MIX

You can specify mixed constants only for maps defined for DBCS devices. DBCS devices are display and printer devices with DBCS capability that enable double-byte character data to be viewed or printed. Double-byte characters are required for languages such as Chinese, Japanese, and Korean.

Mixed constant fields cannot span multiple lines on a DBCS printer.

Target environments for Constant field - MIX

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.

Environment	Compatibility Considerations
MVS batch	None.
IMS/VS	Row 1 column 1 of the map must be blank, or contain a constant or variable field mark character. Unprotected literal constant fields are not supported. The generated Message Format Service (MFS) defines them as though they were protected. If a constant field mark is in the last position of a map (last row, last column), it is treated as though it were in row 1 column 1. A zero-length constant field is not supported for maps. Mix constant fields longer than 255 bytes are not supported.
IMS BMP	If you specify the /MSP=MFS or the /MSP=ALL generation option, then the IMS/VS considerations apply to IMS BMP. Otherwise, there are no compatibility considerations.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	A double-byte character that starts in the last column of a line is split in the middle when displayed on the workstation. To avoid this, do not define a double-byte field that spans lines on the map. Shift-out/Shift-in (SO/SI) characters in a Mixed constant field are converted to blanks when the map is generated in ASCII format.
OS/400	The following compatibility considerations apply to maps containing double-byte fields: • An AS/400 DBCS screen does not display double-byte characters that start in column 80. Instead, a single-byte X is displayed in column 80 and in column 1 of the following line. To avoid this situation, do not define double-byte fields that span lines. • When field outlining is specified for a field on a map, no data other than blanks can be displayed in the first three bytes on a map (row 1, columns 1 through 3). In addition, row 1, column 4 can only be a blank or an attribute byte.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	A double-byte character that starts in the last column of a line is split in the middle when displayed on the workstation. To avoid this, do not define a double-byte field that spans lines on the map. When Shift-out/Shift-in (SO/SI) take position is set, a blank is inserted at the beginning and end of a DBCS portion of the constant.
AIX	Same as OS/2 (C++)
HP-UX	Same as OS/2 (C++)

Environment	Compatibility Considerations
CICS for AIX	Same as OS/2 (C++)
Windows NT	Same as OS/2 (C++)
CICS for Windows NT	Same as OS/2 (C++)
Solaris	Same as OS/2 (C++)
CICS for Solaris	Same as OS/2 (C++)
Test Facility	None.

Field attributes

Field attributes specify display characteristics such as color and highlighting.

Uses

Field attributes are used when displaying maps. Attributes can be specified for each variable and constant field on a map. If the map appears on a device that does not support a specified attribute, that attribute is ignored.

Target environments for Field attribute

See the compatibility considerations for the individual field attributes.

Field attribute - Color

Color specifies the color of the field when it appears on a color device.

Uses

You can specify the following colors:

- Blue
- Green
- Mono (default)
- Pink
- Red
- Turquoise
- White
- Yellow

Definition considerations for Field attribute - Color

The following table shows the field characteristics of the Mono (default) color:

	Normal intensity	Bright intensity
Protected	Blue	White
Unprotected	Green	Red

These were the only colors available on older 3270s that did not support the color attribute. This is called four color mode on newer 3270s to be compatible with older 3270s.

The exception to the above table is when a color attribute appears anywhere on the screen. In this case, four color mode is suppressed and all Mono fields take on the colors of Green (normal) or White (bright). The suppression continues until the screen is cleared, even if the color attribute is overlaid with a Mono attribute without clearing the screen. The program can explicitly clear the screen by coding a `SET <map> PAGE`.

Target environments for Field attribute - Color

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Field attribute - Highlight

Highlight enables you to use different display highlighting techniques.

Uses

You can specify the following highlighting:

No highlight
Text has no special highlighting. This is the default.

Blink Text flashes on and off.

Reverse video
Text and the background colors are reversed. For example, if the display has a dark background with light letters, the background becomes light and the text becomes dark.

Underscore
The field is underlined.

Note: All of these attributes are device dependent and might be supported differently depending on your system configuration.

Target environments for Field attribute - Extended Highlighting

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	An underscore is not supported on color monitors. Using an underscore in maps will cause underscored characters to be displayed using reverse video when generated for the CICS for OS/2 environment.
OS/400	For color devices in the 5250 family, the blink attribute for extended highlighting is supported only for red fields. The blink attribute is ignored for fields of any other color.
OS/2 (GUI)	Not supported.

Field attribute - Highlight

Environment	Compatibility Considerations
Windows (GUI)	Not supported.
OS/2 (C++)	An underscore is not supported on color monitors. Another attribute, such as a variation of reverse video, can be used instead of an underscore.
AIX	Same as OS/2 (C++)
HP-UX	Same as OS/2 (C++)
CICS for AIX	Same as OS/2 (C++)
Windows NT	Same as OS/2 (C++)
CICS for Windows NT	Same as OS/2 (C++)
Solaris	Same as OS/2 (C++)
CICS for Solaris	Same as OS/2 (C++)
Test Facility	Blink highlighting is not supported.

Field attribute - Initial cursor field

Initial cursor field specifies the field in which the cursor is to be positioned when a map is first displayed.

Uses

The default for the initial cursor field is the first named and unprotected variable field in the map.

Initial cursor field removes the previous setting of this attribute from any other fields that might have it set.

Target environments for Field attribute - Initial cursor field

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.

Environment	Compatibility Considerations
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Field attribute - Input required

Input required specifies that the program user must enter data in the field before pressing Enter or an action key.

Note: This is a 3270 attribute that is supported for upward compatibility.

Uses

An error occurs if data is not entered in this field. If any errors occur during the processing of a map and that map is displayed again, then the program user must enter the data in all of the fields that have the Input required attribute specified.

This attribute is simulated by treating it as an input required edit.

By default, the Input required attribute is set to false.

Target environments for Field attribute - Input required

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.

Field attribute - Input required

Environment	Compatibility Considerations
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Field attribute - Intensity

Light intensity specifies the brightness of the value in the field when it displays on the screen.

Uses

You can specify the following intensity:

Normal
Text appears with normal light intensity. This is the default.

Dark Text is not visible. This is useful for passwords.

Bright Text appears with a higher-than-normal light intensity.

Note: All of these attributes are device dependent and might be supported differently depending on your system configuration.

Target environments for Field attribute - Intensity

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Field attribute - Light pen detect

Light pen detect specifies whether a light pen can cause a terminal interrupt for the field.

Note: This is a 3270 attribute that is supported for upward compatibility.

Field attribute - Light pen detect

Uses

A light pen field is usually a variable field.

If a field is light pen detectable, the program is notified when a light pen is pointed at the field (or on some devices, it can be cursor selected).

There are two types of light pen detectable fields:

Immediate detect
Acts as if a device interrupt occurred.

Delayed detect
Sets a detect flag on for a field, but an interrupt does not occur until a function key or the Enter key is pressed.

Definition considerations for Field attribute - Light pen detect

If you specify Light pen detect, consider the following:

- The field must start with certain characters, called designator characters, that determine the action to be taken when the field is selected. See the documentation for the specified devices to determine the correct designator characters.

  The most common designator characters are:

  & Immediate detect

  ? Delayed detect
- On IBM 3278- and 3279-type terminals, either the immediate or delayed detect can be specified. On IBM 3277 terminals, delayed detect is the only type of light pen field that works correctly.
- The field should be protected to prevent the program user from modifying the designator character, and thus changing the effect of selecting the field.
- The field cannot have Dark specified. If you specify Bright for the field, the field must start with the appropriate designator character for the field to be detectable by a light pen.
- The `IF map item MODIFIED` statement can be used to check for any fields that were selected using the light pen.

Target environments for Field attribute - Light pen detect

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.

Environment	Compatibility Considerations
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	The mouse is used in place of the light pen to select a field that is light pen detectable.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	Same as CICS for OS/2.
AIX	Same as CICS for OS/2.
HP-UX	Same as CICS for OS/2.
CICS for AIX	Same as CICS for OS/2.
Windows NT	Same as CICS for OS/2.
CICS for Windows NT	Same as CICS for OS/2.
Solaris	Same as CICS for OS/2.
CICS for Solaris	Same as CICS for OS/2.
Test Facility	None.

Example for Field attribute - Light pen detect

The following is an example of a definition of a variable field using immediate detect:

```
&Update Purchase order file
```

Field attribute - Modified data tag

Modified data tag causes the field to be considered modified when the map first displays.

Uses

Modified data tag enables you to present default data to the program user for that field when the program runs.

Field attribute - Modified data tag

The modified data tag affects only variable fields. When the field appears, the program user can accept the default data by pressing Enter or by typing new data over the default data.

If you do not specify Modified data tag, the program user must change the default data for it to be read by the program.

Definition considerations for Field attribute - Modified data tag

When a map first displays, Modified data tag is set if one of the following occurs:

- Modified data tag is specified as an attribute.
- A `SET map item DEFINED` statement is specified before the map is displayed, and Modified data tag was specified as an attribute.
- A `SET map item MODIFIED` statement is specified in a program before a CONVERSE or DISPLAY I/O option occurs.

Modified data tag remains set until a map appears again. As soon as the map displays, Modified data tag is automatically turned off.

Modified data tag is set on again if one of the following occurs:

- A `SET map item MODIFIED` statement is specified in a program before the display of a map.
- A `SET map CLEAR` statement is specified before a map is displayed and Modified data tag was specified during field attribute definition. A `SET map CLEAR` resets all the fields on a map to their original definition.
- A `SET map item DEFINED` statement is specified before the map is displayed, and Modified data tag was specified as an attribute.

Target environments for Field attribute - Modified data tag

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported
IMS/VS	None.
IMS BMP	Not supported
CICS for VSE/ESA	None.
VSE batch	Not supported

Environment	Compatibility Considerations
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Field attribute - Numeric

Numeric attribute specifies that the field will only accept numeric data.

Note: This is a 3270 attribute that is supported for upward compatibility.

Uses

At run time, the numeric attribute simulates the behavior of 3270 hardware in rejecting character data entered in a field.

Target environments for Field attribute - Numeric

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.

Field attribute - Numeric

Environment	Compatibility Considerations
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
CICS for AIX	Not supported.
Windows NT	Not supported.
CICS for Windows NT	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
Test Facility	None.

Field attribute - Outlining

Field outlining enables you to draw lines at the edges of fields on DBCS devices.

Uses

Outlining is only valid for DBCS devices.

You can specify the following outlining:

Box Draws a line over the field, under the field, to the left of the field, and to the right of the field.

Over Draws a horizontal line over the text from the beginning field mark to the ending field mark.

Under Draws a horizontal line under the text from the beginning field mark to the ending field mark.

Left Draws a vertical line to the left of the text in the position of the beginning field mark.

Right Draws a vertical line to the right of the text in the position of the ending field mark.

Definition considerations for Field attribute - Outlining

When field outlining is specified, an attribute byte at the beginning of an outlined constant or variable field cannot be in the last column of a printer map. An outlined constant or variable field cannot end in the last column of a printer map.

Target environments for Field attribute - Outlining

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	Outlining on workstation printers is supported by overwriting the line with control characters. Only Over and Left are supported, but others can be simulated: • The Under attribute is simulated by using the Over attribute on the next line. • The Right attribute is simulated by using the Left attribute on the next column. • The Under attribute cannot be simulated on the last line of a page. • The Right attribute cannot be simulated on the last character of a line. To ensure that a map is portable, do not use the Under attribute on the last line of a page, or use the Right attribute on the last character of the line.
OS/400	When field outlining is specified for a field on a map, no data other than blanks can appear in the first 3 bytes. Byte 4 can only be a blank or an attribute byte. Some field outlining might disappear when adjacent partial maps appear. The outlining specified for the second map overlays the outlining specified for the first map on the boundary between the two maps.
OS/2 (GUI)	Not supported.

Field attribute - Outlining

Environment	Compatibility Considerations
Windows (GUI)	Not supported.
OS/2 (C++)	Outlining is supported for print maps, but not for display maps.
AIX	Same as OS/2 (C++)
HP-UX	Same as OS/2 (C++)
CICS for AIX	Same as CICS for OS/2.
Windows NT	Same as OS/2 (C++)
CICS for Windows NT	Same as CICS for OS/2.
Solaris	Same as OS/2 (C++)
CICS for Solaris	Same as CICS for OS/2.
Test Facility	None.

Field attribute - Protection

Protection specifies whether data can be entered in the field.

Uses

You can specify the following protection values:

Unprotected
Enables the program user to enter data in a field. This is the default for variable fields.

Protected
Prevents the program user from entering data in a field.

Autoskip
Causes the cursor to automatically move to the next input field as the program user types. This attribute protects the field for which it is defined and should be placed on a field that follows an input field.

Definition considerations for Field attribute - Protection

When moving through the fields on a map, the cursor and the Tab key operate differently depending on what type of protection you specify for each of the fields:

Unprotected	Protected	Autoskip
Cursor moves to this field as you type.	Cursor moves to this field as you type.	Cursor skips this field as you type.
Cursor moves to this field using the TAB key.	Cursor skips this field using the TAB key.	Cursor skips this field using the TAB key.

Target environments for Field attribute - Protection

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	The **Unprotected** value is ignored on constant fields.

Field attribute - Require fill on input

Require fill on input enforces filling a variable field completely or not at all.

Note: This is a 3270 attribute that is supported for upward compatibility.

Uses

By specifying this attribute, you are not defining the variable field as an input required field.

Field attribute - Require fill on input

If an error occurs during the processing of a map and that map is displayed again, then the program user must enter the data to all the fields that have the Require fill on input attribute specified.

This attribute is simulated by treating it as a minimum input edit with the minimum number of characters equal to its field length.

Target environments for Field attribute - Require fill on input

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Message field - EZEMSG

EZEMSG is an optional variable field on a map that is used to display messages.

Uses

A variable field becomes a message field by having its name specified as EZEMSG. The message field must be a character (CHA) or mixed (MIX) field from 11 to 78 bytes.

Definition considerations for Message field - EZEMSG

If a map variable field edit error is detected, messages from the program or the run-time services message table are displayed in the EZEMSG field on the map. If an EZEMSG field does not exist, the screen is cleared prior to displaying the error message. When the program user presses Enter, the map reappears.

Programs can move messages directly into the EZEMSG field prior to displaying a map.

EZEMSG is set to blanks when the program starts and after a CONVERSE, a DISPLAY, a SET map CLEAR statement, or reconverse of a map following detection of an input edit error.

Target environments for Message field - EZEMSG

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.

Environment	Compatibility Considerations
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field

A variable field on a map is a field whose values and field attributes can be modified by a program.

Uses

Data entered by end users in variable fields can be read by a program.

A variable field appears as an outlined box. A variable field is specified by selecting a Variable part from the Parts Palette and placing it at the desired position on the map presentation area. The variable field mark specifies the starting position of a variable field, but it is not a visible character on the map. The starting field mark of the next field is considered the end mark of the previous field.

Definition considerations for Variable field

Variable fields must be at least 1 byte long and cannot wrap to the top of the map.

Target environments for Variable field

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.

Environment	Compatibility Considerations
IMS/VS	Row 1 column 1 of the map must be blank, or contain a constant or variable field mark character. If the map is displayed using the IMS /FORMAT command, all variable fields in the map are set to blanks instead of to the initial values defined in the map definition.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	The following compatibility considerations apply to maps displayed on devices, including DBCS devices, in the 5250 family: • Row 1, column 1, must either be blank or contain a field attribute byte. • The number of variable fields allowed varies with the control unit to which the display device is attached. If the number is less than or equal to 256, the map can be displayed on any 5250 control unit.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field array

A variable field array (also known as map array) is a set of variable fields on a map referenced by a subscripted variable name, and sharing the same definition and edit characteristics.

Variable field array

Uses

A map array is specified by giving the same name to each field in the array and by specifying a unique index (or subscript) for each field in the array. The array variables can be positioned anywhere on the map and the indexes do not have to be in the same order as the fields appear on the map. All index values between 1 and the number of fields in the array must be used for the array to be valid; thus, no gaps in the index values are allowed.

Definition considerations for Variable field array

All variable fields in a map array must have the same data type, length, and share the same map edit specifications. Hardware attributes might be different for each variable field in the map array. The hardware attributes can be set using the properties window or changed at run time using the SET statement.

Target environments for Variable field array

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	Row 1 column 1 of the map must be blank, or contain a constant or variable field mark character. If the map is displayed using the IMS /FORMAT command, all variable fields in the map are set to blanks instead of to the initial values defined in the map definition.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	The following compatibility considerations apply to maps displayed on devices, including DBCS devices, in the 5250 family: • Row 1, column 1, must either be blank or contain a field attribute byte. • The number of variable fields allowed varies with the control unit to which the display device is attached. If the number is less than or equal to 256, the map can be displayed on any 5250 control unit.
OS/2 (GUI)	Not supported.

Environment	Compatibility Considerations
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field - DBCS

DBCS variable fields are map variables that contain only double-byte characters.

Uses

You can specify DBCS variable fields only for maps defined for DBCS devices.

A DBCS variable field appears as an outlined box. A variable field is specified by selecting a DBCS Variable part from the Parts Palette and placing it at the desired position on the map presentation area. The variable field mark specifies the starting position of a variable field, but it is not a visible character on the map. The starting field mark of the next field is considered the end mark of the previous field.

Definition considerations for Variable field - DBCS

DBCS devices are display or printer devices with DBCS capability that allows double-byte character data to be viewed or printed. Double-byte characters are required for languages such as Chinese, Japanese, and Korean.

For variable DBCS fields on a DBCS printer, wrapping fields are allowed only if the map width is equal to the printer width. If you define a wrapping field, you must start the field in an even-numbered column. DBCS variable fields must be at least 2 bytes long and cannot wrap to the top of the map.

The length of these fields must be an even number. Length is expressed in bytes on maps.

Target environments for Variable field - DBCS

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	Row 1 column 1 of the map must be blank, or contain a constant or variable field mark character. If the map is displayed using the IMS /FORMAT command, all variable fields in the map are set to blanks instead of to the initial values defined in the map definition.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	A double-byte character that starts in the last column of a line is split in the middle when displayed on the workstation. To avoid this, do not define a double-byte field that spans lines on the map.
OS/400	The following compatibility considerations apply to maps containing double-byte fields: • An AS/400 DBCS screen does not display double-byte characters that start in column 80. Instead, a single-byte X is displayed in column 80 and in column 1 of the following line. To avoid this situation, do not define double-byte fields that span lines. • When field outlining is specified for a field on a map, no data other than blanks can be displayed in the first three bytes on a map (row 1, columns 1 through 3). In addition, row 1, column 4 can only be a blank or an attribute byte.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	Same as CICS for OS/2.
AIX	Same as CICS for OS/2.
HP-UX	Same as CICS for OS/2.
CICS for AIX	Same as CICS for OS/2.
Windows NT	Same as CICS for OS/2.
CICS for Windows NT	Same as CICS for OS/2.

Environment	Compatibility Considerations
Solaris	Same as CICS for OS/2.
CICS for Solaris	Same as CICS for OS/2.
Test Facility	None.

Variable field - MIX

MIX variable fields are map variable fields that can contain both double-byte character data and single-byte character data.

Uses

You can specify mixed variable fields only for maps defined for double-byte character set (DBCS) devices.

A mixed variable field appears as an outlined box. A mixed variable field is specified by selecting a MIXED Variable part from the Parts Palette and placing it at the desired position on the map presentation area. The variable field mark specifies the starting position of a variable field, but it is not a visible character on the map. The starting field mark of the next field is considered the end mark of the previous field.

Definition considerations for Variable field - MIX

Mixed variable fields can be specified only for maps defined for DBCS devices. Double-byte character sets are required for languages such as Chinese, Japanese, and Korean.

Mixed variable fields cannot span multiple lines on a DBCS printer.

Specifying a variable field as Mixed during map definition forces the data type to be Mixed in the variable field edit definition. The data type can be changed only during map definition.

Mixed variable fields must be at least 2 bytes long and cannot wrap to the top of the map.

Target environments for Variable field - MIX

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.

Environment	Compatibility Considerations
MVS batch	None.
IMS/VS	Row 1 column 1 of the map must be blank, or contain a constant or variable field mark character. If the map is displayed using the IMS /FORMAT command, all variable fields in the map are set to blanks instead of to the initial values defined in the map definition.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	A double-byte character that starts in the last column of a line is split in the middle when displayed on the workstation. To avoid this, do not define a mixed field that spans lines on the map.
OS/400	The following compatibility considerations apply to maps containing double-byte fields: • An AS/400 DBCS screen does not display double-byte characters that start in column 80. Instead, a single-byte X is displayed in column 80 and in column 1 of the following line. To avoid this situation, do not define double-byte fields that span lines. • When field outlining is specified for a field on a map, no data other than blanks can be displayed in the first three bytes on a map (row 1, columns 1 through 3). In addition, row 1, column 4 can only be a blank or an attribute byte.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	Same as CICS for OS/2.
AIX	Same as CICS for OS/2.
HP-UX	Same as CICS for OS/2.
CICS for AIX	Same as CICS for OS/2.
Windows NT	Same as CICS for OS/2.
CICS for Windows NT	Same as CICS for OS/2.
Solaris	Same as CICS for OS/2.
CICS for Solaris	Same as CICS for OS/2.
Test Facility	None.

Variable field edit

A variable field edit is an edit specification defining the editing performed on map variable fields.

Uses

Editing occurs before the fields appear or are printed, or after they are read from the display.

Input edits include validation of the input data. When data is detected that is not valid, the map is shown to the user along with an error message describing why the data is not valid. The user can enter correct data, or press a bypass edit key to bypass input editing. Output edits specify how the data is to be formatted when shown to the user.

Variable field edit elements are associated with the field name. All fields in a map array share the same edit specifications.

Target environments for Variable field edit

See the compatibility considerations for the individual variable field edits.

Variable field edit - Check SO/SI space

Check SO/SI space determines whether mixed data entered in a field on an ASCII device can be converted to the mainframe SO/SI format and still fit in a field of the same length.

Definition considerations for Variable field edit - Check SO/SI space

Mixed fields require fewer bytes of storage on OS/2 systems because the ASCII DBCS format does not use SO/SI escape characters for delimiting DBCS data.

I/O editing considerations for Variable field edit - Check SO/SI space

If you specify Check SO/SI space and input was entered in the field on an ASCII device, the data in the variable field is checked to ensure that there are enough blank spaces at the end of the field to convert the data to SO/SI format.

During conversion from ASCII to EBCDIC, trailing blanks are deleted from the end of the string to allow room for SO/SI delimiters to be inserted around DBCS substrings.

If the input check fails, the value entered by the program user is truncated at the point at which conversion can occur. The value displays to the program user with a warning message and the cursor pointing to the truncated field.

Variable field edit - Check SO/SI space

An ASCII value that can be converted without truncation must have at least two blanks at the end of the field for each DBCS string within the mixed value.

If Check SO/SI space was not specified, the program user can fill up the input field with mixed data. An attempt to convert the value to SO/SI format might result in nonblank data being truncated on the conversion.

Output editing action

None.

Target environments for Variable field edit - Check SO/SI space

Environment	Compatibility Considerations
VM CMS	Ignored.
VM batch	Not supported.
CICS for MVS/ESA	Ignored.
MVS/TSO	Ignored.
MVS batch	Not supported.
IMS/VS	Ignored.
IMS BMP	Not supported.
CICS for VSE/ESA	Ignored.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field edit - Currency

Currency symbol editing indicates whether the currency symbol is supported in the field.

Uses

If currency symbol editing is supported, when the program user enters data, one currency symbol is accepted preceding or following the field. Currency is only valid for numeric fields.

The currency symbol uses a position in the field and must be considered when the field length is defined.

The default character can be changed by your system administrator using the customization procedures for language-dependent options.

I/O editing considerations for Variable field edit - Currency Symbol

If you specify Currency, one currency symbol is accepted preceding or following the field when data is entered by a user.

The currency symbol is removed before the field is placed in internal storage.

If you do not specify Currency, the data in the variable field is checked to ensure that the program user did not enter a currency symbol in the field.

Output editing action

When a value is displayed in the field, a currency symbol is inserted to the left of the left-most significant digit.

For example, if right-justify is specified, the currency symbol is placed to the left of the left-most significant digit with all of the digits justified to the right.

Target environments for Variable field edit - Currency

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.

Variable field edit - Currency

Environment	Compatibility Considerations
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field edit - Date edit mask

Date edit mask provides a way to specify the format for dates to be entered or displayed in a map.

Uses

You can specify the following date edit mask characters:

D, M, Y
D for day, M for month, Y for year

Separator character
Any nonnumeric, single-byte character except D, M, and Y.

Separator characters must be included when defining the date edit mask, but can be omitted when a date is entered at program run time.

Special characters
SYSGREGRN or SYSJULIAN

Date edit masks can be specified in either Gregorian or Julian formats.

The required length differs for data items and map variable items.

Date edit mask is not available when any of the following occur:

- The data type for the data item is DBCS, Mixed, or Hex.
- The length of the data item or map variable field is not valid.
- You have specified that this data item or map variable field is to have decimal places, a currency symbol, a numeric separator, a sign, or any combination of these edit characteristics.

Date edit mask formats

Valid date edit masks can be one of the following formats:

Short Gregorian

The short version of the Gregorian mask must contain the following parts in any order:

YY 2-digit year

MM 2-digit numeric month

DD 2-digit numeric day of month

The mask parts must be separated by any single-byte nonnumeric character except D, M, or Y.

For example, a mask of YY/MM/DD is used to display the date 96/08/05, August 5, 1996.

Long Gregorian

The long version of the Gregorian mask must contain the following parts in any order:

YYYY 4-digit year

MM 2-digit month

DD 2-digit day of month

The mask parts must be separated by any single-byte nonnumeric character except D, M, or Y.

For example, a mask of YYYY/MM/DD is used to display the date 1996/08/05, August 5, 1996.

Short Julian

The short version of the Julian mask must contain the following parts in any order:

YY 2-digit year

DDD 3-digit numeric day of year

The mask parts must be separated by any single-byte nonnumeric character except D or Y.

For example, a mask of DDD-YY can be used to display the date 218-96, which is August 5, 1996.

Long Julian

The long version of the Julian mask must contain the following parts in any order:

YYYY 4-digit year

DDD 3-digit numeric day of year

The mask parts must be separated by any single-byte nonnumeric character except D or Y.

For example, a mask of DDD-YYYY can be used to display the date 218-1996, which is August 5, 1996.

You can also choose the system default date format by specifying either the SYSGREGRN or SYSJULIAN keyword. Depending on the length of the field, SYSGREGRN and SYSJULIAN apply either the short or long format of the date edit mask. The default date format is defined in the hpt.ini file using the following keys: *gregorianLongDateFormat, gregorianShortDateFormat, julianLongDateFormat, julianShortDateFormat*.

For VisualAge Generator Server for MVS, VSE, and VM, date edit masks associated with SYSGREGRN and SYSJULIAN are defined as installation options. For VisualAge Generator Server, date edit masks associated with SYSGREGRN and SYSJULIAN are defined using environment variables.

Note: For information on the keys and values in the hpt.ini file and VisualAge Generator environment variables, refer to the *VisualAge Generator Installation Guide*.

Length of the Date edit mask for data items

The data item length must be valid for the date edit you specify. The length of the data item can be greater than the length of the date edit mask you specify.

Character data item lengths

If you are specifying a date edit mask for a character data item, the length of the data item must be at least the same as the length of the valid date edit mask. The following are valid lengths for character data items for short dates:

- 8 or greater for Gregorian dates
- 6 or greater for Julian dates

If the date edit mask is MM/DD/YYYY, the length must be at least 10.

Numeric data item lengths

If you are specifying a date edit mask for a numeric data item, the length must be at least the same as the number of digits without the separator characters. The following are valid lengths for numeric data items for short dates:

- 6 or greater for Gregorian dates
- 5 or greater for Julian dates

If the date edit mask is MM/DD/YYYY, the length must be at least 8. If it is YY-DDD, the length must be at least 5.

Length of the Date edit mask for map variable fields

The map variable field length must be valid for the date edit you specify. The length of the map variable field must be the exact number required for the date edit mask.

Map variable field length

The map field length field must match the length of the date edit mask you specify. Valid lengths for map variable fields are as follows:

- Must be 8 or 10 for Gregorian dates
- Must be 6 or 8 for Julian dates

For example, if the date edit mask is MM/DD/YYYY, the length of the variable field must be 10; if it is YY-DDD, the length must be 6.

I/O editing considerations for Variable field edit - Date edit mask

The data in the variable field is checked to ensure that the date was entered in the format specified. The program user does not need to enter the leading zeros for days and months, for example, 8/5/1996 can be entered instead of 08/05/1996.

The program user can omit the separator characters, although all leading zeros must be entered in this case, for example, 120796 must be entered instead of 12796 to get 12/7/96.

When the program user enters valid data from a map, the date is converted from the format specified for the field to internal format. If you have specified other editing options, such as edit routines or range checks for dates in numeric fields, the date edit is done first. That is, the date will be in internal format when the other edits are performed.

The internal format for a numeric Gregorian date is 00YYYYMMDD or 00YYMMDD. The internal format for a numeric Julian date is 0YYYYDDD or 0YYDDD. Internal format for a character date is the same as the system default format, with separator characters included.

Variable field edit - Date edit mask

Once the date has been edited and stored in internal format, it is no longer recognized as a date, but simply as data. For example, if a character date is stored in an 8-byte data item that is moved to a 10-byte data item, the 10-byte data item will be padded on the right with blanks. A 2-digit year will not be converted to a 4-digit year.

Output editing action

The date is converted from its internal format into the date format specified for display on the map. Either 2 or 4 digits are displayed for the year, depending on the format specified.

A leading zero is removed from the date, except when the date begins with the year.

Target environments for Variable field edit - Date edit mask

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.

Environment	Compatibility Considerations
CICS for Solaris	None.
Test Facility	None.

Variable field edit - Decimals

Decimals is the number of positions to the right of a decimal point in numeric variables.

Uses

Decimals can only be specified for numeric variables and have a maximum value of 18.

If this variable field is an element of an array, the number of decimal places applies to all variable fields in the array.

The default decimal point character is a period (.). The default character can be changed by your system administrator using the customization procedures for language-dependent options.

The decimal point character uses a position in the map variable field and must be considered when the field length is defined during map presentation.

I/O editing considerations for Variable field edit - Decimals

If a number other than 0 is specified, one decimal point is accepted anywhere in the field. The data is aligned according to the number of decimal places defined for the field. Insignificant digits are truncated without notifying the user.

The decimal point is removed before the field is placed in internal storage.

If no decimal point is entered, the number is assumed to be an integer.

If decimal places are specified as 0, the data in the field is checked to ensure that the program user did not enter a decimal point in the field.

Output editing action

Data appears with a decimal point character aligned according to the number of decimal places specified.

If decimal places are specified as 0, the number is displayed as an integer.

Variable field edit - Decimals

Target environments for Variable field edit - Decimals

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field edit - Description

Description is a 30-character text string that describes the variable field.

Uses

A description is used for documentation only and does not affect execution. The description can be specified in uppercase, lowercase, or mixed case.

Target environments for Variable field edit - Description

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field edit - Edit error message number

Edit error message number specifies a number that identifies a message in a program message table that will be displayed if the corresponding edit fails.

Uses

You can specify a message number for each of the following types of edits. If you specify an edit error message number, that message should be in the

program message table. If you do not specify your own message number, an error message is automatically provided.

Data type
The message number to display when the program user enters data that is not compatible with the data type defined for the variable field.

The default error message is “Data type error in input -- reenter”.

Edit routine
The message number to display when the program user enters data that fails a modulus check (EZEC10 or EZEC11) or table edit check.

The default error message for a modulus check is “Modulus check error on input--reenter”.

The default error message for a table edit check is “Table edit validity error - reenter”.

Input required
The message number to display when the program user does not enter data in a field for which the input required edit was specified.

The default error message is “No input received for required field - reenter”.

Minimum input
The message number to display when the program user does not enter the minimum number of characters required for a variable field that has the minimum input edit specified.

The default error message is “Input minimum length error in contents - reenter”.

Definition considerations for Edit error message number

If an input edit error is detected and a message number is specified for that type of error, that message is displayed when the map is shown to the program user. If a number is not specified for that type of error, a default error message is displayed.

If EZEMSG is defined on the map, the message is displayed in EZEMSG. Otherwise, the message is displayed on a blank panel.

Target environments for Edit error message number

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.

Environment	Compatibility Considerations
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field edit - Edit routine

Edit routine is the name of a routine or edit table for special editing of data that is entered by the program user in a variable field.

Uses

An edit routine can be one of the following:

- One of the following special function words:
  - Modulus 10 check digit routine (EZEC10)
  - Modulus 11 check digit routine (EZEC11)
- The name of a function used as an edit routine.

If the edit function detects an error, it requests the display of the map by moving a message number (or the value 9999, if no message from the message file is required) to the EZEMNO special function word.

- The name of one of the following types of editing tables:
  - Match valid table
  - Match invalid table
  - Range match valid table

  The table must be defined to the program in the table and additional records list.

An edit routine cannot invoke any CONVERSE function or DISPLAY function that writes to the terminal.

I/O editing considerations for Edit routine

The program starts the edit routines for each variable after all other formatting and all other edit checks are successful.

An edit table or function can be assigned to a map array. If an edit table is specified, each item entered in the array is compared against the table. If you specify a function and data is entered in any item in the array, the function will be processed only once. Code the function to do all editing required on the whole array when it runs. The function can check the modified data tag for each item in the array to determine which items were actually entered by the program user.

Output editing action

None.

Target environments for Edit routine

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.

Environment	Compatibility Considerations
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field edit - Fill character

Fill character is the character used to fill unused map variable field positions on output to a display or printer.

Uses

A fill character can be an alphanumeric character, a blank, or a null (binary zeroes). If you do not enter a fill character, a default is provided based on the type of data:

DBCS, Mixed, and CHA
The default is **N** for null characters. For DBCS data, only blanks and null characters are allowed. For mixed data, only SBCS characters are allowed.

Num, Numc, Pacf, and Bin
The default is blank.

Hex The default is 0.

For fields that are not justified, fill characters are added only on the right.

I/O editing considerations for Fill character

There is no input editing action.

Output editing action

Unused positions of the field are filled with the fill character specified.

Variable field edit - Fill character

Target environments for Fill character

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field edit - Fold

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

Uses

Fold does not occur for numeric fields, DBCS fields, or DBCS data in mixed fields.

Target environments for Variable field edit - Fold

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	For folding to be effective, do not specify UCTRAN in the CICS terminal definition. If UCTRAN is specified, CICS translates all input from the terminal to uppercase before it is passed to the program.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	For folding to be effective, EDIT=ULC must be specified on the TRANSACT macro in the IMS GEN. If EDIT=UC, input data will be folded by IMS. If EDIT names a user-supplied transaction input edit routine, the data is edited by that routine and then edited based on any folding requirements specified for the map.
IMS BMP	Not supported.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	Not supported.
CICS for OS/2	For folding to be effective, do not specify UCTRAN. Folding is performed based on the current code page in effect for OS/2.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	Folding is performed based on the current code page.
AIX	Same as OS/2 (C++).
HP-UX	Same as OS/2 (C++).
CICS for AIX	Same as OS/2 (C++).
Windows NT	Same as OS/2 (C++).
CICS for Windows NT	Same as OS/2 (C++).
Solaris	Same as OS/2 (C++).
CICS for Solaris	Same as OS/2 (C++).
Test Facility	None.

Variable field edit - Hex edit

Hex edit specifies that only hexadecimal digits can be entered in a map variable field.

Uses

The data type of the variable field must be CHA. The data item associated with the map variable field must be CHA or Hex.

I/O editing considerations for Variable field edit - Hex edit

If you specify Hex edit, characters entered in the map field must be from the following set:

```
abcdefABCDEF0123456789
```

When data is entered on the map, trailing blanks are converted to zeros.

If you do not specify Hex edit, the variable field is not checked for hexadecimal characters.

Output editing action

None.

Target environments for Variable field edit - Hex edit

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.

Environment	Compatibility Considerations
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field edit - Input required

Input required specifies that valid data must be entered in a map field.

Uses

The input required edit is satisfied if both of the following are true:

- The field contains a value other than blanks, or zero for a numeric field.
- The program user entered the data in the field or the program set the modified attributed on for the field before the map was conversed.

Blanks, or a zero in a numeric field, will not satisfy the input-required edit check. If blanks or zeros are valid values, and you want to ensure sure that the program user typed data in the field, use the Minimum input edit.

I/O editing considerations for Variable field edit - Input required

If you specify Input required, the data in the variable field is checked to ensure sure that valid data is in the field.

Output editing action

None.

Target environments for Variable field edit - Input required

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.

Variable field edit - Input required

Environment	Compatibility Considerations
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field edit - Justify

Justify specifies the position of data in a variable field when the data is shorter than the length of the field.

Uses

You can specify one of the following justify values:

Left Data is aligned to the left of the field.

Right Data is aligned to the right of the field.

None No justification.

If a justification is not specified, character data is left-justified and numeric data is right-justified. Right-justification is required for numeric fields that also have a decimal position or a sign edit specified.

I/O editing considerations for Variable field edit - Justify

Numeric data is always right-justified and zero-filled. Character data is aligned as specified.

Output editing action

The value is positioned based on the specified justification.

Target environments for Variable field edit - Justify

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field edit - Maximum value

Maximum value specifies the largest number a program user can enter in a variable field.

Variable field edit - Maximum value

Uses

If you specify the maximum value, you must also specify the minimum value, otherwise zero is assumed to be the minimum value.

I/O editing considerations for Variable field edit - Maximum value

The data entered in the variable field is checked to ensure that it is less than or equal to the value specified for Maximum value.

Output editing action

None.

Target environments for Variable field edit - Maximum value

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field edit - Minimum input

Minimum input specifies the minimum number of characters the program user must enter in the variable field. The default is 0.

Uses

To test whether a program user typed data into a field, the program can set the modified data tag on with the minimum input value specified as 1. When the modified data tag is not on for the field, no check is made.

I/O editing considerations for Variable field edit - Minimum input

When the modified data tag is on for the field, the data in the variable field is checked to ensure that the minimum number of characters have been entered.

Output editing action

None.

Target environments for Variable field edit - Minimum input

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.

Variable field edit - Minimum input

Environment	Compatibility Considerations
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field edit - Minimum value

Minimum value specifies the smallest number a program user can enter in a variable field.

Uses

If you specify the minimum value, you must also specify the maximum value.

I/O editing considerations for Variable field edit - Minimum value

The data in the variable field is checked to ensure that it is greater than or equal to the value specified for Minimum value.

Output editing action

None.

Target environments for Variable field edit - Minimum value

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.

Environment	Compatibility Considerations
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field edit - Numeric separator

Numeric separator specifies that data containing numeric separators can be entered in a variable field.

Uses

Numeric separators use positions in the field and must be considered when the field length is specified.

The default numeric separator is a comma (,). The default character can be changed by your system administrator using the customization procedures for language-dependent options.

If the number of significant digits is fewer than 4, Separator is not valid.

Note: You cannot specify Separator with date edits.

I/O editing considerations for Variable field edit - Numeric separator

If you specify Separator, numeric separators are allowed in the field when a program user enters data.

The numeric separators are removed before the field is placed in internal storage.

If you did not specify Separator, the data in the variable field is checked to ensure that the program user did not enter a numeric separator in the field.

Output editing action

When a value is displayed in the field, numeric separators are inserted between every 3 significant digits; every fourth position to the left of the decimal point is a separator.

Variable field edit - Numeric separator

Target environments for Variable field edit - Numeric separator

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field edit - Sign

Sign specifies whether a sign should appear in a field and whether it is a leading or trailing sign.

Uses

Signs can only be specified for numeric fields.

You can specify one of the following sign values:

Leading
Accept on input or display a plus (+) or a minus (–) sign to the left of numeric data.

If you specify Leading, you must also specify Right for the Justify edit.

Trailing
Accept on input or display a plus (+) or a minus (–) sign to the right of numeric data.

If you specify Trailing, you must also specify Right for the Justify edit.

None Ensures that a sign is not entered in the field.

The sign uses a position in the field and must be considered when the field length is specified.

I/O editing considerations for Variable field edit - Sign

Leading
A plus (+) or a minus (–) must be entered on input to the left of a numeric data item.

Trailing
A plus (+) or a minus (–) must be entered on input to the right of a numeric data item.

None Validation is done to ensure that the program user did not enter a sign anywhere in the field.

Output editing action

If the field is not large enough to hold both the sign and the value of the number, a positive sign is omitted. If the value is negative, the numeric value is truncated on the left to display the negative sign. To ensure that the sign and the entire number is always visible, define the field length to be at least one greater than the length of any numeric item moved to the map field.

Leading
A sign displays to the left of the left-most significant digit. Positive leading signs (+) are not displayed. If a leading sign and currency symbol are specified, the sign precedes the currency symbol when data appear.

Trailing
A sign displays to the right of the number.

None The number displays without a sign, even if the number is negative.

Variable field edit - Sign

Target environments for Variable field edit - Sign

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field edit - Zero edit

Zero edit specifies how zero values are displayed in numeric fields.

Uses

If you specify Zero edit, a zero value is displayed as the number zero. If you do not specify Zero edit, a zero value is displayed as if it were a character field containing blanks.

The display format depends on the other edit characteristics specified for the variable field.

Note: Other editing characteristics like decimal positions, currency symbol, and numeric separator are also applied to the variable field.

How Zero Edit Affects Edits

If you specify Zero edit, the following rules apply:

- If the fill character is 0, the data is formatted with the character 0.
- If the fill character is nulls, the data is left-justified.
- If the fill character is blanks, the data is right-justified.
- If the fill character is an asterisk (*), the asterisk is used as a filler instead of a blank.

How Zero Edit Affects Variable fields

The following table shows a list of the contents of a numeric field when Zero edit is specified and when Zero edit is not specified. The sample field is defined as right-justified with a length of 11. A "b" represents a blank fill character.

Decimal places (2)	Currency or Separator	Fill Character	Zero edit not selected	Zero edit selected
no	no	N	nulls	0
no	no	b	blanks	bbbbbbbbbb0
no	no	0	00000000000	00000000000
no	no	*	**********	*********0
yes	no	N	nulls	0.00
yes	no	b	blanks	bbbbbbb0.00
yes	no	0	00000000000	00000000.00
yes	no	*	**********	*******0.00
yes	yes	N	nulls	$0.00
yes	yes	b	blanks	bbbbbb$0.00
yes	yes	0	00000000000	$000,000.00
yes	yes	*	**********	******$0.00

I/O editing considerations for Variable field edit - Zero edit

There is no input editing action.

Output editing action

Depending on whether or not you specify Zero edit, editing is done on numeric fields with a value of 0 to transform them into the specific format

described in the table. If there is a value of zero to the left of the decimal point, one significant zero is displayed in front of the decimal point.

If the zero value was entered with a negative sign (–), the negative sign appears if both the zero edit option and a sign are specified.

Target environments for Variable field edit - Zero edit

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field edit order

Variable field edit order specifies the order in which variable field edits take place when the program is running or being tested.

Definition considerations for Variable field edit order

The default order is determined by the position of the variables on the map, from left to right and top to bottom. A different order can be specified by reordering the edit order graphical tags. Regardless of the specified order, the edit routines are called after all other formatting and checking is done. First, all other edits are performed for each field in the order specified by the edit order tags. Then the edit routine specified for each field is started in the order specified by the edit order tags.

For array elements, the edits are performed in the order of their indices.

Target environments for Variable field edit order

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	None.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field length

Variable field length specifies the number of bytes between the beginning variable field mark and the ending variable field mark.

Uses

The length of the variable field includes the places set aside for decimal point, sign, currency symbol, and numeric separators.

Definition considerations for Variable field length

A maximum of 18 digits can be displayed in a numeric variable field. However, a variable field can be longer than 18 bytes if numeric edits are specified, such as decimal point, sign, currency symbol, and numeric separators.

When the map width does not equal the device width, a variable field cannot wrap to the next line. DBCS and mixed variable fields also place restrictions on the length of variable fields.

Target environments for Variable field length

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.

Environment	Compatibility Considerations
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Variable field name

Variable field name is an identifier for a map variable field.

Definition considerations for Variable field name

The naming restrictions for a variable field name are the same as for a data item. For further details, see Appendix B. Naming conventions for data item, record, function names.

Target environments for Variable field name

Environment	Compatibility Considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.

Environment	Compatibility Considerations
HP-UX	None.
CICS for AIX	None.
Windows NT	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Part 2. Scripting language

Chapter 10. Program processing statements

Statement Elements

Table 17. Program Statement Elements

Element	COBOL											GUI		C++								Java	TEST FACILITY
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	Windows NT	
AID value	x		x	c		c		x		c	x			x	x	x	c	x	c	x		x	
Assignment statement	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		x	
CALL statement	c	c	c	c	c	c	c	c	c	c	c	x	x	c	c	c	c	c	c	c		c	
Data item	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		x	
DXFR statement	c	c	c	c	c	c	c	c	c	c	c			c	c	c	c	c	c	x		c	
FIND statement	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		x	
IF statement	x	x	x	x	x	c	x	x	x	c	x	c	c	x	x	x	x	x	x	c		c	
I/O error value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		x	
MOVE statement	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		x	
MOVEA statement	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		x	
RETR statement	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		x	
SET statement	x	x	x	x	x	x	x	x	x	x	x	c	c	x	x	x	x	x	x	x		x	
SYS value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		x	
TEST statement	x	x	x	x	x	c	x	x	x	x	x	c	c	x	x	x	x	x	x	c		x	

Table 17. Program Statement Elements (continued)

Element	COBOL											GUI		C++								Java	TEST FACILITY
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	Windows NT	
WHILE statement	x	x	x	x	x	c	x	x	x	c	x	c	c	x	x	x	x	x	x	c		c	
XFER statement	c	c	c	c	c	c	c	c		c	c			c	c	c	c	c	c	x		c	

Note: *Includes Windows 95 and Windows 98, Windows NT, and Windows 2000.

Legend: In this table, the following characters are used to indicate the level of support:
x Supported
c Supported with compatibility considerations
blank Not supported

AID value

AID value is a syntactical element common to many VisualAge Generator statements. AID values are used to test the state of the EZEAID special function word. EZEAID identifies which interrupt key the program user pressed.

Attribute	Description
BYPASS	Any of the keys specified as bypass keys for the map, or for the program, if none were specified for the map.
ENTER	The ENTER key was pressed.
PA	Any PA key was pressed.
PAn	Where “n” is an integer from 1 to 3. PAn is on if the PA key with the corresponding number was pressed.

Attribute	Description
PF	Any function key was pressed.
PFn	Where “n” is an integer from 1 to 24. PFn is on if the function key with the corresponding number was pressed.

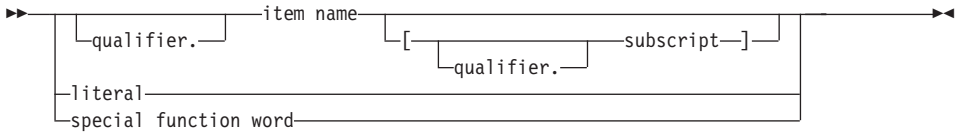
Target environments for AID value

Environment	Compatibility considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	PF6 and PA3 are reserved for a panel recovery function. If the program user presses PF6 or PA3, it is treated as pressing the Clear key. The AID value is not passed back to the program.
MVS batch	Not supported.
IMS/VS	PA1, PA2, and PA3 are reserved for paging by IMS/VS. If your installation uses PF12 for the IMS local copy function, PF12 cannot be used. If these keys are pressed, no AID value is passed to the program.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	The mapping of the personal computer keyboard keys to 3270 keys is defined in the CICS OS/2 Workstation Setup (WSU) table. The VisualAge Generator test facility supports the default CICS OS/2 mapping. The default mapping is: **Program Function Key** — **Personal Computer Key** PF1 - PF12 — F1 - F12 PF13 - PF24 — Alt+F1 - Alt+F12 PA1 - PA3 — Ctrl+F1 - Ctrl+F3 Refer to the *CICS OS/2 System and Application Guide* for information on how to modify the WSU table to change the key mapping. Closing the Map Monitor during CONVERSE is the same as pressing PA2 which is the default bypass edit key. Thus, after closing the Map Monitor during a CONVERSE, EZEAID will have a value of ’PA2’.
OS/400	None.
OS/2 (GUI)	Not supported.

Environment	Compatibility considerations
Windows (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	The mapping of the AIX terminal keyboard keys to 3270 keys is defined in the 3270keys file, which is part of TCP/IP. The default values depend on what type of terminal or session you are using to run CICS for AIX transactions. For further information, refer to the section ″3270keys File Format for TCP/IP″ using InfoExplorer, in the AIX online help facility.
Windows NT (C++)	None.
Windows NT (Java)	None.
CICS for Windows NT	The mapping of the Windows NT (C++) terminal keyboard keys to 3270 keys is defined in the 3270keys file, which is part of TCP/IP. The default values depend on what type of terminal or session you are using to run CICS for Windows NT transactions. For further information, refer to the section ″3270keys File Format for TCP/IP″ using InfoExplorer, the Windows NT (C++) online help facility.
Solaris	None.
CICS for Solaris	The mapping of the Solaris terminal keyboard keys to 3270 keys is defined in the 3270keys file, which is part of TCP/IP. The default values depend on what type of terminal or session you are using to run CICS for Solaris transactions. For further information, refer to the section ″3270keys File Format for TCP/IP″ using InfoExplorer, in the Solaris online help facility.
Test Facility	None.

Data item

Data item is a syntactical element common to many VisualAge Generator statements. A data item identifies a record data item, table column, or map variable that is used as the source or recipient of a data value in a statement.



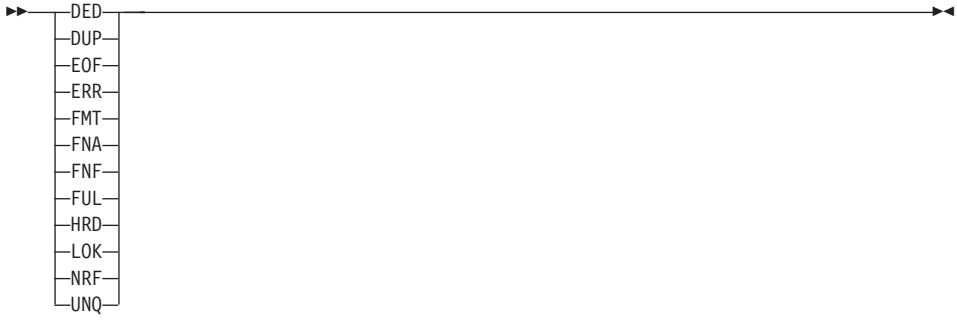
Attribute	Description
qualifier.	The name of the record, table, or map that contains the item being referenced. If a qualifier is not supplied and the statement is in a function, the data item is resolved from: 1. The item names in the local storage and parameter lists 2. One and only one of: • The function object and the items of the function object • The records and the items of the records in the parameter list • The records and the items of the records in the local storage list If the name is not unique across this category, the result is an ambiguous item reference message. 3. One and only one of: • Program working storage and the items of the program working storage • Table and additional records list and the items of the tables and records in the table and additional records list • Called parameter list and the items of the records and maps in the called parameter list • Other I/O objects and the items of other I/O objects in the program If the name is not unique across this category, the result is an ambiguous item reference message. If the item is not resolved and you have specified in program specification to allow implicits, the generator creates an implicit definition for the item based on how it is used within the program. Implicit data item names are not supported in functions that use parameters, local storage or a return value. Qualified subscript names are allowed to help avoid name conflicts. Items that appear in more than one structure must be qualified either explicitly or implicitly.
item name	The name of a record data item, table column, or map variable field.

Attribute	Description
subscript	When a data item represents an array of values (the item represents a map array, a table column, or a record data item with multiple occurrences), the subscript identifies which element in the array is accessed. The subscript must be a numeric literal or the name of a data item that contains an integer value. If the subscript is omitted, the first element in the array is accessed. The subscript name may be qualified. A subscript can be implicit.
literal	Constant values can be specified in place of a data item name when the data item is used as the source of a value in a statement. There are four types of literals: character constant, numeric, mixed, and DBCS. A character constant is a string of characters enclosed in quotes. If single quotes (') are used, the string is folded to uppercase. If double quotes (") are used, the literal is used as entered. A number is not enclosed in quotes. The number can have a leading sign and can include a decimal. A mixed single-byte and double-byte literal is enclosed in quotes. On System/370 a DBCS literal has the form `G'<XXXX>'` where: < represents the shift-out (SO) character > represents the shift-in (SI) character XXXX represents one or more DBCS characters The SO/SI characters are deleted when the DBCS value is moved to a DBCS data item. **Note:** In OS/2, DBCS and MIX literals are entered the same as on System/370, except without SO/SI characters. For example, the System/370 literal `G'<XXXX>'` is the same as the OS/2 DBCS literal `G'XXXX'`. Refer to *VisualAge Generator Design Guide* for more information.
special function word	Special function words are predefined names that can be specified as operands in statements. All special function words begin with the prefix EZE, such as EZEDAY, EZEDEST, or EZEMSG.

I/O error value

I/O error values indicate the type of error, if any, that occurred on the last access to a file or database record. The error values that can be set depend on the record organization.

Note: When using ODBC, both the SQLCODE and SQLSTATE fields are set in EZESQLCA. However, the value in SQLCODE is unreliable. It is highly recommended that you instead use the I/O error value to test the SQL record for error states.



Attribute	Description
DED	Tests whether a deadlock occurred when two separate transactions attempted to updated records being held by other locks. DED only occurs for SQL row records and is a hard error. When using DB2, the SQLCODE is -911. The DED error code can be received only if EZEFEC is equal to 1.
DUP	Tests for a duplicate key. The I/O option might or might not have been successful, depending on whether duplicate keys are allowed.
EOF	Tests for end of a file.
ERR	Tests for a return code other than zero (0) for file and relational databases, or blank for DL/I databases.
FMT	Tests the format of a file against that which has been defined.
FNA	Tests for the availability of a file.
FNF	Tests for “no file found” for the record.
FUL	Tests for a full file or temporary storage queue key greater than 32767. FUL is not set for non-VSAM serial files in MVS/TSO and MVS batch. Instead, an abnormal end (B37) is received.
HRD	Tests for any hard I/O error.
LOK	Tests for a lockout condition on an OS/400 system.
NRF	Tests for “no record found” in a file or database.
UNQ	Tests whether an attempt was made to add or replace a record in a file or database for which a duplicate key already exists.

Uses

VisualAge Generator distinguishes between two classes of I/O return codes during execution. They are classified as hard or soft errors. The program continues processing on soft errors if the user provides an error routine for the I/O function. The hard errors cause the program to be terminated with

error messages unless the program has set the EZEFEC switch to 1, indicating that the program will handle hard I/O errors, and an error routine is defined for the function.

Abend codes are errors that cause the program to terminate. These cannot be processed by the program under any circumstances.

Return code results differ in meaning depending upon the type of file organization used.

DED Tests for the following:

- Whether a deadlock occurred when two separate transactions attempted to updated records being held by other locks.

  DED only occurs for SQL row records and is a hard error. When using DB2, the SQLCODE is -911. The DED error code can be received only if EZEFEC is equal to 1.

DUP Tests whether a duplicate key already exists. The I/O option might or might not have been successful, depending on whether duplicate keys are allowed. DUP is set in the following situations:

- For an indexed, relative, or serial file, DUP is a soft error and is set for the following:
  - When using an ADD I/O option, an attempt is made to add a record to a file, or to add an entry to an alternate index with a record ID (key) that duplicates another record ID that already exists in the file or alternate index. If duplicate IDs (keys) are allowed when the file is defined, the record is added to the file. Otherwise, the record is not added to the file. DUP might not indicate failure, depending on the data set or index being added or updated.

    Note: The DUP mnemonic is returned only if the access method returns this information. Therefore, adding a duplicate record to the file might return the DUP mnemonic on some operating systems, but not on others.
  - When using a REPLACE I/O option, an attempt is made to replace a record in a file while a record in an alternate index has the same key. If duplicate keys are permitted, the replace is successful. DUP might not indicate failure depending on the data set or index being added or updated.
  - When using the SCAN, SCANBACK, INQUIRY, or UPDATE I/O options, a record is successfully read. However, records exist in the file with keys that were duplicates of the key field of the record returned.

- For a DL/I database, DUP is a soft error. DUP is set when an attempt is made to add a record to a database in which records with duplicate keys are not permitted and a record with the same key already exists in the database. The ADD failed. The status code is II.
- For an SQL row record, DUP is a hard error. DUP is set when an attempt is made to add or replace a row in a database. The value being inserted for the column is a duplicate value. DUP always indicates the add or replace failed. For DB2, the SQLCODE is -803. This mnemonic can only be received if EZEFEC is 1.

  DUP and UNQ are not identical for SQL row records. DUP and UNQ are both set when the DUP condition occurs and a unique index is defined for one of the SQL columns. Only DUP is set when the DUP condition occurs and a unique index is not defined for any of the SQL columns.
- For OS/400, see Resource Source Association tag `:FILE /DUP` option in *VisualAge Generator Generation Guide*.

EOF Tests whether the end of a file has been reached. EOF is a soft error. End of file occurs when:

- For a serial or relative file, the last record in the file was accessed by the previous retrieval of a record (end of file).
- For an indexed file:
  - The last record in the file was accessed by the previous retrieval of a record (end of file).
  - A SET record SCAN specified a position that had no records following it in the file.
  - For SCANBACK, the first record of a file was accessed by the previous retrieval of a record (top of file) or SET record SCAN specified a position that had no records preceding it in the file.
  - For Non-CICS environments:
    - if empty - ERR, EOF, EZERT8=102
    - if uninitialized - ERR, EOF, NRF, EZERT8=205
  - For CICS environments:
    - if empty and SCAN - ERR, EOF, NRF, EZERT8=102
    - if empty and SCANBACK - ERR, EOF, EZERT8=102
    - if uninitialized and SCAN - ERR, EOF, NRF, EZERT8=102
    - if uninitialized and SCANBACK - ERR, EOF, EZERT8=102
- For a serial file implemented as a GSAM file, no more segments exist in the database. The status code is GB.
- For a serial file implemented as an IMS message queue, no more messages exist on the message queue. The status code is QC.

- For a DL/I database, the end of the database has been reached. The program was positioned at the end of the database and a SCAN was requested. The status code is GB.

ERR Tests for the following:

- For a serial, indexed, or relative file, a nonzero return code was received from the I/O operation.
- For a serial file implemented as a GSAM file, a nonblank status code was returned by DL/I.
- For a serial file implemented as an IMS message queue, a nonblank status code was returned by DL/I.
- For a DL/I database, a nonblank status code was returned by DL/I or a nonzero condition code was returned in response to the DL/I call.
- When using DB2 for an SQL row record, the SQLCODE was anything other than 0.

Note: ERR might be either a hard or soft error, depending on the type of error.

FMT Tests whether the format of the file associated with the record matches the record definition. FMT can occur for any I/O option. This is a hard error that indicates that the record I/O operation failed.

To test for FMT, the value of the special function word EZEFEC must be equal to 1.

Some conditions that would cause a format (FMT) error to occur are as follows:

Record Length
For fixed-length records, the record length of the file is not equal to the record length of the VisualAge Generator record. For variable-length records, the record length of the file is larger than the record length of the VisualAge Generator record.

Record Format
The format of the VisualAge Generator record (fixed or variable length) does not match the format of the file.

Key Length
The key length of the indexed VisualAge Generator record does not match the key length of the indexed file.

Key Offset
The key offset of the indexed VisualAge Generator record does not match the key offset of the indexed file.

Record Organization
The organization of the VisualAge Generator record (serial, indexed, or relative) does not match the organization of the file.

File Type
The file type specified for the VisualAge Generator record, such as SEQ, SEQRS, VSAM, or VSAMRS, does not match the file type of the file.

FNA Tests whether the file associated with the record is available. FNA can occur for any I/O option.

Test for FNA when another program could be using the file or when system resources for accessing the file might be scarce. This is a hard error that indicates the record I/O operation failed.

To test for FNA, the value of the special function word EZEFEC must be equal to 1.

FNF Tests whether the file associated with the record can be found. FNF can occur for any I/O option. This is a hard error that indicates the record I/O operation failed.

To test for FNF, the value of the special function word EZEFEC must be equal to 1.

FUL Tests if a file is full. This is a hard error that indicates the record I/O operation failed.

To test for FUL, the value of the special function word EZEFEC must be equal to 1.

FUL is set in the following situations:

- For a serial or indexed file because the file was full.
- For a serial or relative file implemented as a CICS temporary storage queue, an add attempted to insert a key greater than 32767.

HRD Tests for any hard I/O error. A hard error is an I/O operation that is not successful.

The following are not considered hard errors: EOF, NRF, and LOK. DUP and UNQ are not considered hard errors for most record types; however, they are hard errors for SQL row records.

The following are considered hard errors: FUL, FMT, FNF, FNA, and DED.

To test for HRD, the value of the special function word EZEFEC must be equal to 1.

To test for HRD after DL/I database operations, either the value of the special function word EZEDLERR must be equal to 1 or the value of the special function word EZEFEC must be equal to 1.

HRD is set in the following situations:

- For a serial, indexed, or relative file, any file I/O error other than an error defined as a soft error. Soft errors set EOF, NRF, DUP, UNQ, or LOK.
- For a serial file implemented as a GSAM file, any nonblank status code returned by DL/I. The status code is not GB.
- For a serial file implemented as an IMS message queue, any nonblank status code returned by DL/I. The status code can be any status code other than QC, QD, CE, CF, CG, CI, CJ, CK, or CL.
- For a DL/I database, any nonblank DL/I status code or nonzero CICS DL/I error code. The DL/I status code can be any status code other than GA, GB, GD, GE, GK, or II.
- When using DB2 for an SQL row record, the SQLCODE is 304, 802, or less than 0.

LOK Tests whether a lockout condition exists on an OS/400 system.

For serial, indexed, or relative files, lockout occurs when two separate transactions attempt to update the same record in a file. It also occurs when you attempt to delete or replace a record that is not locked for update.

LOK is a soft error.

NRF Tests whether a "no record found" condition exists during record I/O operations. NRF is always a soft error. NRF is set in the following situations:

- For an indexed file:
  - On an INQUIRY or UPDATE I/O option, no record is found for the specified record ID
  - On a SCANBACK for an empty file in the CICS environment
- For a relative file:
  - On an INQUIRY or UPDATE I/O option, no record is found for the specified record ID.
  - For a SCAN, when scanning beyond the end of the file
- For a serial file implemented as an IMS message queue, no more message segments exist for the last message being read from the message queue. The status code is QD.
- For a DL/I database, no record is found in the database that satisfies the selection conditions specified in the DL/I call. This

state can be set for an ADD I/O option if the parent of a segmented to be inserted is not found. The status code is GE.

- For an SQL row record:
  - On an INQUIRY or UPDATE I/O option, no row is found in the database that satisfies the selection conditions specified in the SELECT statement.
  - For a SCAN, there are no rows left that were selected for scanning.

    When using DB2, the SQLCODE is 100 in both situations.

UNQ Tests if an attempt was made to add or replace a record in a file or database with a key that already exists. If the UNQ condition exists, the ADD or REPLACE failed. UNQ is set in the following situations:

- For an indexed, relative, or serial file, UNQ is a soft error. UNQ is set when:
  - When using an ADD I/O option, an attempt is made to add a record to a file or an entry into an alternate index with a record ID (key) that duplicates another record ID that already exists in the file or alternate index. The file definition indicates that duplicate keys are not allowed.
  - When using a REPLACE I/O option, an attempt is made to replace a record in a file in which duplicate keys are not permitted, and a record with the same key already exists. The key can be an alternate index key.
- For a DL/I database, UNQ is a soft error. UNQ is set when an attempt is made to add a record to a database in which records with duplicate keys are not permitted and a record with the same key already exists in the database. The status code is II. UNQ is equivalent to DUP for DL/I records.
- For an SQL row record, UNQ is a hard error. UNQ is set when an attempt is made to add or replace a row in a database for which one of the columns being replaced has a unique index defined. The value being inserted for the index column is a duplicate value. When using DB2, the SQLCODE is -803. This mnemonic can only be received if EZEFEC is 1.

  DUP and UNQ are equivalent for SQL row records. DUP and UNQ are both set when the DUP condition occurs and a unique index is defined for one of the SQL columns. Only DUP is set when the DUP condition occurs and a unique index is not defined for any of the SQL columns.

I/O status codes

EZERT8 contains the file I/O status code. Use the /SYSCODES generation option to control the codes that are returned for file I/O errors. The /SYSCODES generation option value does not affect the use of VisualAge Generator mnemonics.

- If /NOSYSCODES is specified, EZERT8 contains system **independent** codes.
- If /SYSCODES is specified, EZERT8 contains system **dependent** access method return codes. For VisualAge Generator, that is the COBOL file status key value. Refer to the COBOL reference information for your environment for information on the COBOL File Status Key values.

The following table describes the correspondence between status key values, mnemonics, and EZERT8 in COBOL environments. There is a many-to-1 correspondence between the file status key values and EZERT8.

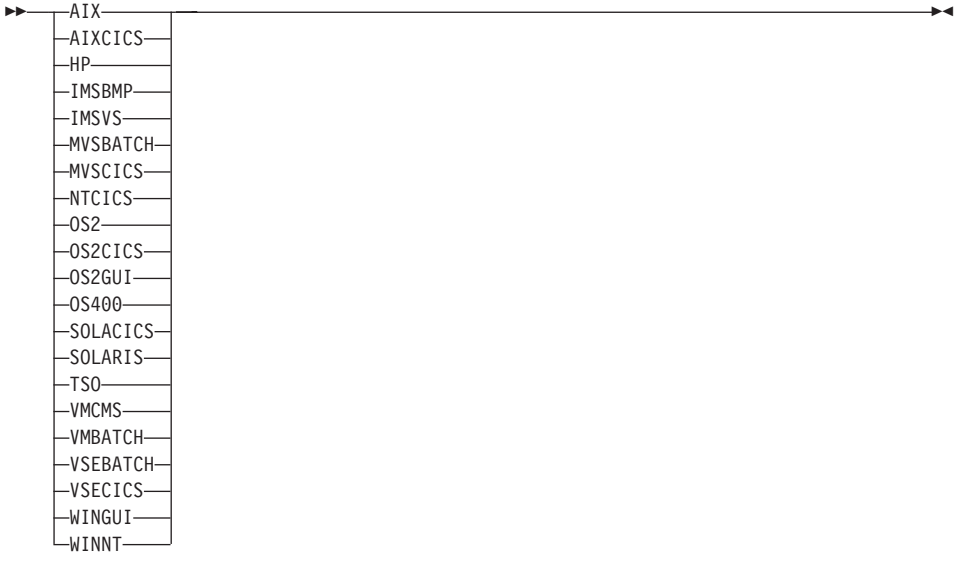
EZERT8 - /SYSCODES, COBOL File Status Key values	**VisualAge Generator Mnemonics**	**EZERT8 - /NOSYSCODES**
00,05,07	NORMAL	000
02	DUP, ERR	103
04 (Var record format)	NORMAL	000
04	FMT, ERR, HRD	220
10,14,46	EOF, ERR	102
22	UNQ, ERR	206
23 (START)	EOF, ERR	102
23	NRF, ERR	205
24,34 (access method not relative or relative key not 0)	FUL, ERR, HRD	25A
35	FNF, ERR, HRD	251
38	FNA, ERR, HRD	218
39,95	FMT, ERR	220
9D (OS/400 only)	LOK, ERR, HRD	381

For all other file status codes, EZERT8 in COBOL environments is set based on the type of request as shown in the following table:

Type of Request	VisualAge Generator Mnemonics	EZERT8 - /NOSYSCODES
OPEN	ERR, HRD	500
CLOSE, UNLOCK	ERR, HRD	989
READ, START	ERR, HRD	987
WRITE	ERR, HRD	988

SYS value

SYS values are used to test the state of the EZESYS special function word. EZESYS identifies the system on which the program is running.



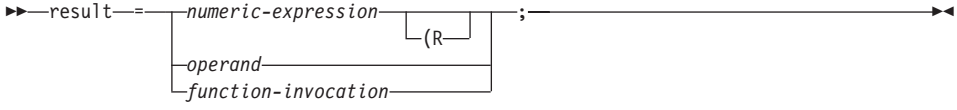
Attribute	Description
AIX	Tests true if the program is running in the AIX environment.
AIXCICS	Tests true if the program is running in the CICS for AIX environment.
HP	Tests true if the program is running in the HP-UX environment.
IMSBMP	Tests true if the program is running in the IMS BMP environment.
IMSVS	Tests true if the program is running in the IMS/VS environment.
MVSBATCH	Tests true if the program is running in the MVS batch environment.
MVSCICS	Tests true if the program is running in the CICS for MVS/ESA environment.

SYS value

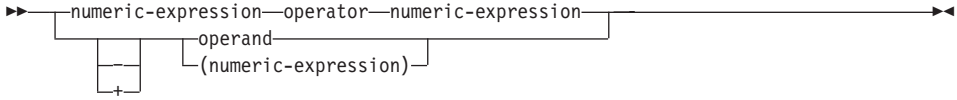
Attribute	Description
NTCICS	Tests true if the program is running in the CICS for Windows NT (C++) environment.
OS2	Tests true if the program is running in the OS/2 (C++) environment.
OS2CICS	Tests true if the program is running in the CICS for OS/2 environment.
OS2GUI	Tests true if the program is running in the OS/2 GUI environment.
OS400	Tests true if the program is running in the OS/400 environment.
SOLACICS	Tests true if the program is running in the CICS for Solaris environment.
SOLARIS	Tests true if the program is running in the Solaris environment.
TSO	Tests true if the program is running in the MVS/TSO environment.
VMCMS	Tests true if the program is running in the VM CMS environment.
VMBATCH	Tests true if the program is running in the VM batch environment.
VSEBATCH	Tests true if the program is running in the VSE batch environment.
VSECICS	Tests true if the program is running in the CICS for VSE/ESA environment.
WINGUI	Tests true if the program is running in the Windows environment.
WINNT	Tests true if the program is running in the Windows NT (C++) environment.

Assignment statement

The Assignment statement specifies arithmetic calculation or data movement. You can use the assignment statement to assign the value of an arithmetic expression to a data item, assign the value of one item to another, or assign the values of items in one structure to corresponding items (items with the same names) in another structure.



numeric expression



Attribute	Description
result	The result for an arithmetic assignment statement can be a data item, EZETST, EZEMNO, or EZERCODE. If it is a data item, it can be subscripted, qualified, or both, and it must be numeric, packed, or binary. For other assignment statements, the result can be a data item, record, map, or certain special function words as listed in the table on Table 20 on page 451. If the result is a data item, it can be subscripted, qualified, or both. For Assignment statement examples, see "Examples for assignment" on page 383.
function invocation	The operand can be a function invocation of a user defined function or of certain VAGen supplied special function words. A function used in an assignment statement must be defined to have a return value that is compatible with the result of this statement. For the general syntax of function invocations, see "Function invocation statement" on page 400 or for details of invoking a specific VAGen supplied function, refer to the description of that word in "Special function words" on page 451.
operator	The following arithmetic operators are supported: + Operands are added. – The second operand is subtracted from the first operand. * Operands are multiplied. / The first operand is divided by the second operand. // The result is the remainder of a division of the first operand by the second operand. The remainder operator cannot be used with other operators. All operators, except unary operators, must be surrounded by a space or line boundary, except when preceded or followed by parenthesis. No operators are allowed on assignment statements involving records, maps, functions, or nonnumeric data.

Attribute	Description
numeric expression	When the expression is an arithmetic calculation the following true: • The expression can consist of 1 to 255 operands. Up to 500 characters can be entered using the Assignment Statement Template. • The number of implicits in an arithmetic statement is limited to 16. • The arithmetic operators can be any of the five binary operators (+, –, *, /, //). These operators require two operands. • The arithmetic operators can be one of the two unary operators (+, –). These operators precede their operand. • Parentheses can be used to specify the order in which the arithmetic expression is evaluated. When an arithmetic expression contains nested parentheses, the nested expression is evaluated before the expression in which it is contained. The number of opening and closing parentheses must match, except for the rounding option symbol, (R. A closing parenthesis is not allowed for the rounding option symbol. If you specify the rounding option, you must code it after the last closing parenthesis. • When you do not use parentheses, the following order of evaluation applies: – Unary operators are performed first. – Multiplication and division are performed next. – Addition and subtraction are performed last. – Within a statement, the operations are performed according to priority as they are encountered in the statement from left to right. All first priority operations are performed before any second priority operations. **Note:** Do not use the remainder operator with other arithmetic operators or with the rounding option. For Assignment statement examples, see “Examples for assignment” on page 383.

Attribute	Description
operand	For arithmetic assignment statements, the operand can be a data item, numeric literal, or any numeric special function word. If it is a data item, it can be subscripted, qualified, or both, and must be numeric, packed, or binary. For other assignment statements, the operand can be a data item, literal, record, map, or certain special function words as listed in the table on Table 20 on page 451. If the operand is a data item, it can be subscripted, qualified, or both. **Note:** For all assignment statements where the operand is a numeric literal, you must surround the numeric literal with at least one blank. Any record data item can be specified as an argument on string function calls. In addition, numeric literals can be specified on calls to some string function words. See "String Function EZE words" to determine whether numeric literals can be used with a particular string function.
(R	For arithmetic assignment statements, use (R to round the results after all arithmetic calculations are completed. If you do not specify the rounding option, the result is truncated. **Note:** The rounding option cannot be used with the remainder operator. Although the maximum supported length is 18, the maximum length of a data item used as a result of the rounding option is 17. This is because one digit is added internally to the original data item precision when performing the rounding option. Any violations for numeric and packed decimal items are detected at preprocessing time. No detection occurs for binary and numeric map field items if more than 17 digits are used in the rounded arithmetic calculation. Instead, overflow occurs during execution. Numeric implicits are created with a length of 17 instead of 18 when used as result data items of a rounded operation. If you define an arithmetic statement with more than one operation, all intermediate operations are carried out without the rounding option. Only the result of the last operation is rounded. This is done by adding five to the digit at precision one higher than the precision of the result and then truncating. For Assignment statement examples, see "Examples for assignment" on page 383.

Attribute	Description
function invocation	The operand can be a function invocation of a user defined function or of certain VAGen supplied special function words. A function used in an assignment statement must be defined to have a return value that is compatible with the result of this statement. For the general syntax of function invocations, see "Function invocation statement" on page 400 or for details of invoking a specific VAGen supplied function, refer to the description of that word in "Special function words" on page 451.

Achieving consistent results across environments

Due to truncation of intermediate results, COBOL programs might have different results than GUI or C++ programs for the same arithmetic statements.

To ensure consistent results across environments, use only one binary operator per statement. Multiple addition and subtraction operators can be safely combined if the number of decimal places defined for the result item is greater than or equal to the number of decimal places in any of the operands.

The remainder operator can produce inconsistent results if the result or any of the operands are defined with decimal places greater than zero. To get a consistent remainder with decimal places, use the following algorithm instead of the remainder operator:

```
 quotient = dividend / divisor ;
remainder = dividend - (quotient * divisor) ;
```

Overflow conditions

You can test and control overflow conditions resulting from arithmetic calculations using EZEOVER and EZEOVERS special function words.

Compatibility with CSP/AE arithmetic

Customers moving programs from CSP/AE to VisualAge Generator can specify `/MATH=CSPAE` when generating the programs for host COBOL environments to ensure that the results of arithmetic expressions are the same in COBOL as they were when running under CSP/AE. If standard COBOL arithmetic is satisfactory, use the default option, `/MATH=COBOL`, instead for better performance.

Compatibility with CSP/AE is not supported in the test facility, in GUI, or in generated C++ programs.

CSP/AE statements that follow the guidelines for compatibility in the previous section provide consistent results in all environments.

Target environments for assignment

Supported in all environments without compatibility considerations.

Examples for assignment

The following examples use the Assignment statement as an arithmetic expression or a MOVE statement:

An arithmetic expression with parentheses

The following example shows an arithmetic expression that uses parentheses.

```
PERCENT-CHANGE = (NEW-VALUE - OLD-VALUE) * 100 / OLD-VALUE;
```

The processing order in the above example is determined by the parenthesis and the precedence of operators, as follows:

```
1st   intermediate result1 = (NEW-VALUE minus OLD-VALUE)
2nd   intermediate result2 = intermediate result1 multiplied by 100
3rd   PERCENT-CHANGE = intermediate result2 divided by OLD-VALUE
```

An assignment statement to move or initialize data

When the assignment statement involves data movement and the source expression consists of one operand, the assignment statement works exactly like the MOVE statement.

The following example shows how you use the assignment statement to move data or initialize data:

```
OLD-VALUE = NEW-VALUE;
```

Example of valid arithmetic statements

The following are valid arithmetic statement examples:

```
OP1 = OP2 + OP3 * OP4;     /* Need space around *, /, +, -, //
OP1 = OP2 * (OP3 + OP4);   /* Parentheses force addition first.
OP1 = OP2 - -OP3;          /* There can be blanks between the unary
OP1 = OP2 + -OP3;          /*   sign and the operand.
OP1 = OP2 - -(OP + OP);    /* Unary sign before parenthesis is valid.
OP1 = OP2 + -(-OP3);       /* OP3 is an operand, (-OP3) is an
                           /*   operand.
OP1 = -OP2 + OP3;          /* You can start with unary minus.
OP1 = OP2 + OP3[R];        /* [R] is a subscript.
OP1 = (OP2) (R;            /* You can round a single operand.
OP1 = OP2 /(OP3 + OP4);    /* No space needed between / and (.
```

Example of arithmetic statements that are not valid

The following arithmetic statement examples are **not** valid:

```
OP1 = OPERAND1 - - -OPERAND2;  /* Two consecutive signs not allowed.
OP1 = OP2+OP3*OP4;             /* Need spaces around + and *.
OP1 = OP2 (- OP3);             /* Missing operator.
OP1 = OP2 *(OP3 +(OP4 / OP5);  /* Unmatched parentheses.
OP1 = OP2 + OP3 * OP5 [R];     /* No space allowed before subscript,
```

Example of valid assignment statements

The following are example assignment statements:

```
MESSAGE_FIELD = 'Enter option';   /* Set up message.
MESSAGE-NUMBER = 007;             /* Initialize message number.
INIT-MAP = CUST-INFO-RECORD;      /* Initialize map fields.
```

Example assignment statement that is not valid

The following assignment statement is **not** valid:

```
MESSAGE_FIELD = ('Enter option'); /* Do not use ()s with character data
```

Rounded arithmetic statement with multiple operations

The following example uses the /MATH=CSPAE option:

```
RESULT = OP1 + OP2 + OP3 (R;
```

Where:

```
Field     Decimal
Name     Places  Length  Value

RESULT    2         4
OP1       4         5       1.2345
OP2       3         4       5.678
OP3       4         4        .1169
```

This statement is executed as follows:

```
HOLD = OP1 + OP2 = 1.234 + 5.678 = 6.912
RESULT = HOLD + OP3 = 6.912 + .116 = 7.028
RESULT = 7.028 + 0.005 = 7.033
RESULT = 7.03
```

Without rounding, the statement is executed as follows:

```
HOLD = OP1 + OP2 = 1.23 + 5.67 = 6.90
RESULT = HOLD + OP3 = 6.90 + .11 = 7.01
```

Note: Truncation occurs on the operands to match the characteristics of the RESULT field.

Arithmetic statement with a negative number

The following example uses the /MATH=CSPAE option.

```
Without rounding:
A = B - C;

Where: A has 2 decimal places and a length of 3

B = 1.111
C = 3.888

A = -2.78
```

Note: If the number is negative, rounding is applied to the absolute value.

When you use the rounding option, a variable overflow condition can occur, depending on the value and the defined number of characters of the result item.

Arithmetic statement with a variable overflow

The following example uses the /MATH=CSPAE option:

```
Without rounding:
   A = B + C;

Where:  A has 2 decimal places and a length of 3
   B = 8.888
   C = 1.111

   A = 9.99 with no overflow


With rounding:
   A = B + C (R;

Where:  A has 2 decimal places and a length of 3
   B = 8.888
   C = 1.111

   A = 0.00 with overflow
```

The possibility of maximum value overflow increases because the operand value increments.

Arithmetic statement with division with a remainder

The following example uses the /MATH=CSPAE option.

Use integers in the operand and result portion of the statement when dividing for remainder. To obtain remainders when dealing with numbers other than integers, use the length and number of decimal places for the result for the remainder.

The following formula is used when calculating the remainder:

```
REMAINDER = DIVIDEND - (DIVISOR * QUOTIENT);
```

Note: In the preceding formula, the value in the QUOTIENT has the same length and number of decimal places as the REMAINDER, as illustrated in the following example:

```
  Name       Type   Length   Dec

REMAINDER1   NUM     8        3
REMAINDER2   NUM     8        0

A. REMAINDER1 = 12345 // 10000 = 5.000
   QUOTIENT1 = 12345 / 10000 = 1.234
```

Example A is calculated as follows:

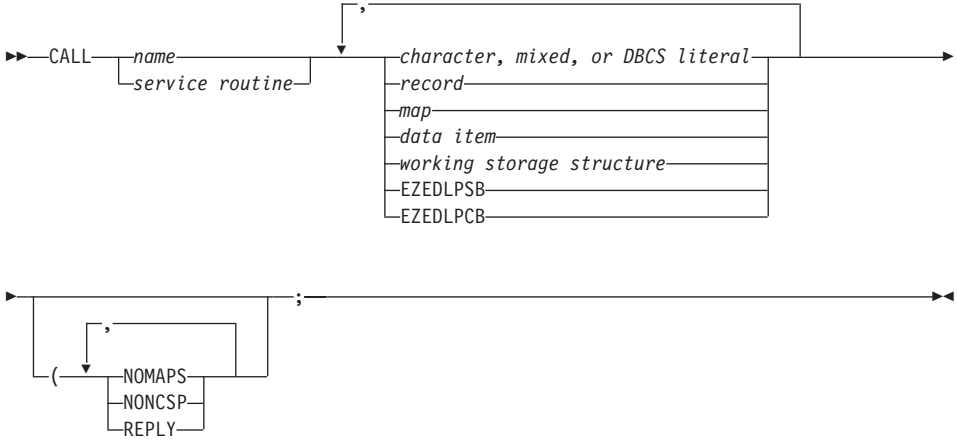
```
   REMAINDER1 = 12345 - (10000 * 1.234) = 5.000


B. REMAINDER2 = 12345 // 10000 = 2345
   QUOTIENT2 = 12345 / 10000 = 1
```

Example B is calculated as follows:

```
   REMAINDER2 = 12345 - (10000 * 1) = 2345
```

CALL statement

CALL transfers control to another program or non-VisualAge Generator program. When the called program or program ends, the current function continues with the statement following the CALL.



Attribute	Description
name	The name of a VisualAge Generator called program, or the name of a non-VisualAge Generator program. For more information on calling non-VisualAge Generator programs, refer to the section on transferring program control in the *VisualAge Generator Client/Server Communications Guide* manual.
service routine	Service routines: AUDIT, COMMIT, CREATX, CSPTDLI, or RESET

Attribute	Description
argument	A character, mixed, or DBCS literal, record, map, data item, working storage structure, EZEDLPSB, or EZEDLPCB. An argument name can be qualified and/or subscripted. If working storage is passed, the level-77 items within it are not passed unless explicitly included in the list of arguments. The maximum number of arguments supported is 30. The arguments specified must match the parameters defined for the called program.

Attribute	Description
option	For logic parts used within a GUI client, the NOMAPS on a CALL statement is ignored. The option can be one or all of the following: **NOMAPS** Use NOMAPS on CALL statements in main or called transaction programs to indicate that the called program or program does not use the screen. Maps displayed before the CALL statement do not need to be refreshed. If you do not specify NOMAPS, the screen is refreshed when control is returned to the calling program. If your program converses a full-screen map after the CALL, use the NOMAPS option even if the called program might have used the screen. This prevents the refreshing of the screen prior to the output of a map that completely overlays the prior map. **Note:** Although this recommendation applies to calls to VisualAge Generator programs, it can also be done for calls to non-VisualAge Generator programs as long as the calling program does not CONVERSE the same full-screen map before and after the call. **REPLY** Indicates the status of a call by setting a return code in EZERT8. REPLY is effective with any of the service routines in the CICS environments, with calls to remote server programs. **Note:** In the IMS environments, the REPLY option is only effective when used with CREATX. The return code indicates why the service or remote call could not be initiated and is available in the EZERT8 special function word as the 8-character displayable form of the return code. REPLY cannot be used to test the program return code from a called program. Program return codes should be passed back using a parameter. If REPLY was not specified or is not effective and an error is detected in the service, the calling program ends with an error message that explains the reason for the termination and displays the return code. Termination for GUI clients means that a walkback is generated.

Attribute	Description
option (continued)	**NONCSP** Indicates that the target program is a non-VisualAge Generator program. When used with the CALL statement, the NONCSP option affects the passing of parameters to CICS programs defined with REMOTE or DYNAMIC linkage in the linkage table. You can use it to improve performance in the test facility as it indicates that the called program or program is not to be run from the MSL. You can also define the characteristics of the called program in the linkage table as an alternative to coding the NONCSP option on each CALL. Refer to *VisualAge Generator Client/Server Communications Guide* for more information.

Definition considerations for CALL

Whenever data is passed to another program, modification of that data by the called program or program effectively modifies the storage of the calling program. Recursive calls (A calls A; or A calls B, which calls A) are not supported, except with C++ generated programs.

The type of linkage used on the CALL and the format of the parameters passed during generation or test execution varies by system. The default linkage for generated programs is described in "Target environments for CALL" on page 390.

You can use the linkage table to request that other types of linkage be generated for calls to specific programs. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on transferring program control, preparing programs for generation, and for more information regarding the linkage table.

Calls to remote called batch programs

At program generation, use the linkage table to specify that a CALL is a call to a remote called batch program (a called batch program that is generated to receive CALLs from a remote system).

The linkage table describes the type of linkage to be generated for both called and calling programs, including how the location of the called program is identified, and what kind of data format conversion needs to be performed on the call.

The input and output arguments on a remote call are passed and returned by value, not by pointer. Arguments that overlap in storage (same argument passed more than once or multiple definitions of the same record) cannot be passed on a remote call. The total number of bytes in the data structures defined for the arguments must be less than 32567 bytes.

The REPLY option allows continuation with a nonzero system error code from the remote CALL function. The return code is available to the program in EZERT8 if REPLY is specified.

Refer to the section on implementing client/server processing using the CALL statement in *VisualAge Generator Client/Server Communications Guide* for more information on using the CALL statement for calling programs on remote systems.

Target environments for CALL

Any record item (not just a level 77 item) can be an argument on a string function.

Environment	Compatibility Considerations
VM CMS	When default linkage is used, a CALL is implemented as a dynamic COBOL CALL. Register 1 points to the parameter list. The return code set in Register 15 by the called program is not passed back to the calling program. Calling a remote program is not supported. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on transferring program control.
VM batch	Same as VM CMS.

Environment	Compatibility Considerations
CICS for MVS/ESA	When default linkage is used in CICS environments, a CALL is implemented using a CICS LINK. The parameter list is passed in the CICS COMMAREA. The return code set in register 15 by the called program is not passed back to the calling program. When using the COMMDATA linkage convention on a CICS/ESA system, the maximum COMMAREA length (total bytes of all parameters passed) is 32763. The maximum COMMAREA length is 32763 for a call to a remote program. Calling a remote program is supported only in CICS/ESA Version 3 Release 3 or later systems. If EZELOC is used to specify the target system for a remote CALL, the generated program will not precompile correctly on earlier systems. Calling a remote called batch program running on the same system is supported, allowing programs on a host system and programs on workstations to share the same called program running on the host. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on transferring program control.
MVS/TSO	When calling a non-VisualAge Generator program OS LINK is used. Otherwise, same as VM CMS.
MVS batch	Same as MVS/TSO.
IMS/VS	If the initial program is a main transaction, a CALL to a batch program that accesses the I/O PCB as a serial file is not supported. When default linkage is used, a CALL is implemented as a dynamic COBOL CALL. Register 1 points to the parameter list. The return code set in register 15 by the called program is not passed back to the calling program. When calling a non-VisualAge Generator program OS LINK is used. Calling a remote program is not supported. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on transferring program control. The REPLY option is only effective when used with CREATX.
IMS BMP	The REPLY option is only effective when used with CREATX. Otherwise, same as MVS/TSO.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	Same as VM CMS.

Environment	Compatibility Considerations
CICS for OS/2	When default linkage is used in CICS environments, a CALL is implemented using a CICS LINK. The parameter list is passed in the CICS COMMAREA. The return code set in register 15 by the called program is not passed back to the calling program. The maximum COMMAREA length is 32567 for a call to a remote program. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on transferring program control. If a program calls a remote called batch program that accesses DL/I databases, the calling program must pass EZEDLPSB as a parameter to the remote called batch program. The PSB is scheduled in the first remote program and the CICS UIB address is passed back in the EZEDLPSB parameter for use on subsequent calls to remote DL/I programs. In a single logical unit of work, all calls to DL/I remote called batch programs must go to the same target system.
OS/400	If the program is a non-VisualAge Generator program, the CALL uses the standard OS/400 CALL interface. Parameters are passed using a standard system argument list.
OS/2 (GUI)	None.
Windows (GUI)	None.
OS/2 (C++)	Recursive calls are supported. A linkage table entry is only needed for the calling programs. Called programs do not require a linkage table entry. Calls to remote programs are supported as well as calls to existing local or remote CICS programs. Refer to the *VisualAge Generator Generation Guide* document for more information on defining linkage tables. The REPLY option is supported on calls to remote programs.
AIX	Same as OS/2 (C++)
HP-UX	Same as OS/2 (C++)
CICS for AIX	Recursive calls are supported. Default linkage is via a CICS statement passing pointers in the CICS COMMAREA The generation linkage table can be used to request that parameters be passed by value in the COMMAREA or to specify that the called program is a remote server program. EZEDLPSB must be passed as a parameter to a remote server program if the server program accesses DL/I databases and the unit of work extends across multiple server calls.
Windows NT (C++)	Same as OS/2 (C++).

Environment	Compatibility Considerations
Windows NT (Java)	Same as OS/2 (C++).
CICS for Windows NT	Same as CICS for AIX.
Solaris	Same as OS/2 (C++)
CICS for Solaris	Recursive calls are supported. Default linkage is via a CICS statement passing pointers in the CICS COMMAREA The generation linkage table can be used to request that parameters be passed by value in the COMMAREA or to specify that the called program is a remote server program. EZEDLPSB must be passed as a parameter to a remote server program if the server program accesses DL/I databases and the unit of work extends across multiple server calls.
Test Facility	The REPLY option can be used on calls to COMMIT and RESET. The test facility displays a message that corresponds to the value set in EZERT8. Recursive programs are not supported.

Examples for CALL

To go to another program called APPL2, passing a data item called ITEM1, enter the following:

```
CALL APPL2 ITEM1;
```

or:

```
CALL APPL2 ITEM1 (NOMAPS;
```

The parenthesis preceding the option is required.

DXFR statement

DXFR transfers control to another program or program. The current program ends and any open files are closed.

Attribute	Description
program	Name of the program to be initiated.

Attribute	Description
EZEAPP	Special function word used to dynamically specify the program name on a DXFR statement. This special function word enables you to change the transferred-to program name within a program.
record	Name of any record used in the current program. The information in the record is used to initialize the working storage record of the transferred-to program. The data in the record must be compatible with the record expected by the transferred-to program. Compatible working storage must be defined for the program that is the object of the transfer. If a working storage record is specified on an DXFR, only the data in the structure is transferred. Any level-77 data items defined are not transferred. If the receiving working storage is not the same size as the one transferred, the smaller size is used for the transfer. If the receiving area is larger, the primary working storage record of the transferred-to program is initialized based on the type of data (blanks for character data, and zero for numeric data). The initialization is done before the transferred record, if any, is moved into the primary working storage record. If the definition of the record specified on the DXFR statement is not compatible with the definition of the primary working storage record of the transferred-to program, unpredictable results, including abnormal termination or the display locking up, can occur in the transferred-to program. For example, the following conditions might cause incompatibilities between the two records to occur: • The records differ in length • The field boundaries of the two records do not correspond • The type of data differs (for example, the field is defined as character in the record used on the DXFR statement, but the transferred-to program expects the field to be DBCS data).
NONCSP	Indicates that the target program is a non-VisualAge Generator program. The option can be specified on the DXFR statement or in the linkage table on a :DXFRLINK statement. The option is required when generating the program for a COBOL environment. Transfers to VisualAge Generator and non-VisualAge Generator programs are generated differently in those environments. If the NONCSP option is specified, the implementation of the DXFR depends upon the environment. See “Target environments for DXFR” on page 395 for more information.

Definition considerations for DXFR

You can specify either the name of the program or the special function word, EZEAPP. EZEAPP enables you to dynamically change the transferred-to program name in a program.

DXFR is designed to transfer control to another program, but to stay in the same CICS or IMS/VS transaction. For environments that do not support transactions, DXFR provides similar function within the same run unit.

DXFR cannot be used from a called program. You cannot transfer using a DXFR to a main transaction that has a first map defined. If the transfer of control is to another program, it must be defined as main transaction or main batch.

Generation considerations for DXFR

A linkage table entry specifying static linkage for non-CICS MVS host environments and the target program is a generated program that calls PL/I programs and the programs are not using LE/370.

Target environments for DXFR

Environment	Compatibility Considerations
VM CMS	Transfer to non-VisualAge Generator programs is done using the OS XCTL macro. The record is passed as a parameter. For DXFRs to non-VisualAge Generator programs, the maximum record size is 32757. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on transferring program control. If you are using generated programs as saved segments, the following restrictions apply: • If the saved segment is an initial program then you cannot transfer with a DXFR statement that uses an XCTL or XFER to other programs • You cannot transfer with a DXFR statement that uses an XCTL or XFER to a program that is loaded as a saved segment
VM batch	Same as VM CMS.
CICS for MVS/ESA	The DXFR is implemented using the CICS XCTL command for both VisualAge Generator programs and non-VisualAge Generator programs. The record is passed using the COMMAREA option of XCTL. The data starts in the first byte of the common area. The maximum record size is 32763. In CICS environments only, a commit occurs on a DXFR when a PSB is scheduled at the time of the DXFR. You can use the /NOSYNCDXFR generation option to prevent a commit in CICS when the transferring and transferred-to program use the same PSB (have the same PSB part name specified in the program definition). Refer to the section on generation options in the *VisualAge Generator Generation Guide* document for details. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on transferring program control.

Environment	Compatibility Considerations
MVS/TSO	Transfer to non-VisualAge Generator programs is done using the OS XCTL macro. The record and EZEDLPSB (if a PSB was used in the transferring program) are passed as parameters. For DXFRs to non-VisualAge Generator programs, the maximum record size is 32757. All programs in the same run unit must share the same DL/I PSB. The PSB part definition can vary if EZEDLPCB is used to pass information on the CALL. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on transferring program control.
MVS batch	DL/I calls and GSAM files are supported in the transferred-to program only if there was a PSB specified for the transferring program and the transferring program does at least one of the following: • Uses CSPTDLI • Associates at least one file or EZEPRINT with GSAM • Uses EZEDLPSB or EZEDLPCB in any statement in the program • Has DL/I databases other than ELAWORK or ELAMSG in the PSB definition Otherwise, the same as MVS/TSO.
IMS/VS	A transfer using DXFR to a non-VisualAge Generator program is not supported. The NONCSP option is ignored. DXFRLINK with LINKTYPE=NONCSP in the linkage table is not supported. Programs that run under the same transaction using DXFR must share the same PSB, must have the same execution mode, and must use the IMS scratchpad area (SPA) in the same way. A commit point never occurs at DXFR. SPA use is specified as a generation option. If the initial program in a transaction is a main batch program, DXFR to a main transaction program is not supported. If the initial program is a main transaction, DXFR to a main batch program that accesses the I/O PCB as a serial file is not supported.
IMS BMP	Same as MVS/TSO.
CICS for VSE/ESA	Same as CICS for MVS/ESA.

Environment	Compatibility Considerations
VSE batch	DXFR is not supported for non-VisualAge Generator programs. DL/I CALL files are supported in the transferred-to program only if there was a PSB specified for the transferring program and the transferring program does at least one of the following: • Uses CSPTDLI. • Uses EZEDLPSB or EZEDLPCB in any statement in the program • Has DL/I databases other than ELAWORK or ELAMSG in the PSB definition All programs in the same run unit must share the same PSB. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on transferring program control.
CICS for OS/2	Same as CICS for MVS/ESA.
OS/400	Control is passed directly to the program to be initiated using the OS/400 XCTL interface. Working storage is passed as a parameter using a standard system argument list. The program issuing the DXFR is removed from the program invocation stack and does not resume control when the initiated program ends.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	The DosExecPgm API is used to transfer control to a non-VisualAge Generator program. The record is passed via a transfer block in shared memory. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on how to transfer control from a VisualAge Generator program to a non-VisualAge Generator program.
AIX	The exec() and fork() system calls are used to transfer control to a non-VisualAge Generator program. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on how to transfer control from a VisualAge Generator program to a non-VisualAge Generator program.
HP-UX	The exec() and fork() system calls are used to transfer control to a non-VisualAge Generator program. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on how to transfer control from a VisualAge Generator program to a non-VisualAge Generator program.
CICS for AIX	The DXFR is implemented using the CICS XCTL command. The record is passed using the COMMAREA option of XCTL. The data starts in the first byte of the COMMAREA. The maximum record size is 32763.
Windows NT (C++)	Same as OS/2 (C++).

Environment	Compatibility Considerations
Windows NT (Java)	DXFR may only be used for UI records and local Java Server Programs.
CICS for Windows NT	Same as CICS for AIX.
Solaris	The exec() and fork() system calls are used to transfer control to a non-VisualAge Generator program. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on how to transfer control from a VisualAge Generator program to a non-VisualAge Generator program.
CICS for Solaris	The DXFR is implemented using the CICS XCTL command. The record is passed using the COMMAREA option of XCTL. The data starts in the first byte of the COMMAREA. The maximum record size is 32763.
Test Facility	None.

Examples for DXFR

To transfer control and pass the record MYRECD to the program called NEWAPP1, type:

```
DXFR NEWAPP1 MYRECD;
```

To transfer control to another program called APPL2, type:

```
DXFR APPL2;
```

To use EZEAPP to specify a variable name, type:

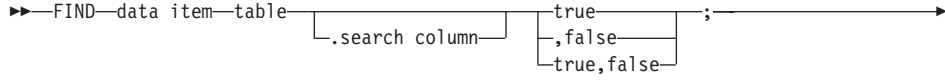
```
MOVE 'APPL2' TO EZEAPP;
DXFR EZEAPP;
```

To pass a working storage record to a non-VisualAge Generator program, type:

```
MOVE 'APPL3' to EZEAPP;
DXFR EZEAPP COMMON_DATA_RECORD (NONCSP;
```

FIND statement

FIND transfers control to a function depending on whether a data item value is found in the search column of a table.



Attribute	Description
data item	Name of a data item or literal. The data item name can be subscripted, qualified, or both.
table	Name of a table to be searched.
search column	Name of column in a table to be searched. The default is the first column in a table.
true	Name of a main function, or EZECLOS, if specified in a program flow. The name of a function, EZECLOS, EZEFLO, or EZERTN if specified in a function.
false	Name of a main function or EZECLOS, if specified in a program flow. The name of a function, EZECLOS, EZEFLO, or EZERTN if specified in a function.

If the data item value is found in the specified column in any of the rows in the table, the true element is executed as an unconditional statement. If the true element is not specified, execution continues with the statement following the FIND.

If the data item value is not found in the specified column in any of the rows in the table, the false element is executed as an unconditional statement. If the false element is not specified, execution continues with the statement following the FIND.

The special function word EZETST is loaded with a value depending on the results of a FIND statement. EZETST contains the following values:
- Zero if no match is found
- The row number of the first row where a match is found. If the data in the search column is not unique, the first occurrence in the table is the one used.

When EZETST contains a row number, it can be used as a subscript for other statements that reference other columns in that same row of the table.

For more efficient performance, use FIND instead of IF...OR....OR.

In the **statements area**, if you specify both a true and a false name on a FIND statement, you can separate the names with a comma. If you specify only a false name, you must precede it with a comma. Commas are automatically inserted for you if you are using the FIND statement template.

Target environments for FIND

Supported in all environments without compatibility considerations.

Examples for FIND

In the following example, a table (INFO) has 50 rows and 3 columns called STATE, POPULTN, and AREA. The first column has an entry for each of the 50 states, the second column contains the population for each state, and the third column contains the area in square miles for each state.

INFO:

STATE	POPULATN	AREA
Alabama	4,041,000	51,600
Alaska	550,000	586,000
.	.	.
.	.	.

A FIND statement could be used to pick up the row number containing a certain state from the table above and branch accordingly.

```
MOVE 'ALASKA' TO ITEM;
FIND ITEM INFO.STATE MATCH,NOMATCH;
```

When the FIND is executed and a match is found, EZETST is set to 2. This is the row number of the matching state. Control is passed to MATCH. If no match is found, EZETST is set to 0 and control is passed to NOMATCH.

The following statement will pass control to MATCH only if a match is found.

```
FIND ITEM INFO.STATE MATCH;
```

If no match is found, the statement immediately following the FIND statement is run.

The following statement will pass control to NOMATCH if a match is not found.

```
FIND ITEM INFO.STATE ,NOMATCH;
```

If a match was found, the statement immediately following the FIND statement is run.

Function invocation statement

Function transfers control to another function.

Attribute	Description
function	The name of a function.

Definition considerations for Function invocation statement

When the function invocation statement ends, control is returned to the statement following the function invocation statement.

Function invocation statements do not use flow statements. If a function is used both as a main function and as a function invocation statement in the same program, the flow statements will not be executed when the function is performed.

Function invocation statements cannot be used as edit routines for map items. However, they can be performed from map variable field edit routines. DISPLAY or CONVERSE functions cannot be performed from a map edit group either directly or from any functions started during map edit group execution.

Note: Do not use function invocation statements for unconditional flow, transfer, or return processing. Use the special function word EZERTN for an immediate return to the invoking function. Use the special function word EZEFLO for "go to" or transfer processing.

Any record data item can be specified as an argument on string function calls. In addition, numeric literals can be specified on calls to some string function words. See "String Function EZE words" to determine whether numeric literals can be used with a particular string function.

Target environments for function invocation statements

Supported in all environments without compatibility considerations.

Examplesof function invocation statements

To pass a record as an argument and return the result to ARESULT:

```
TESTIT-WSREC.ARESULT = RFUNCMAX(TESTIT-WSREC);
```

To pass data items as arguments and return the result to BIGGESTNUM:

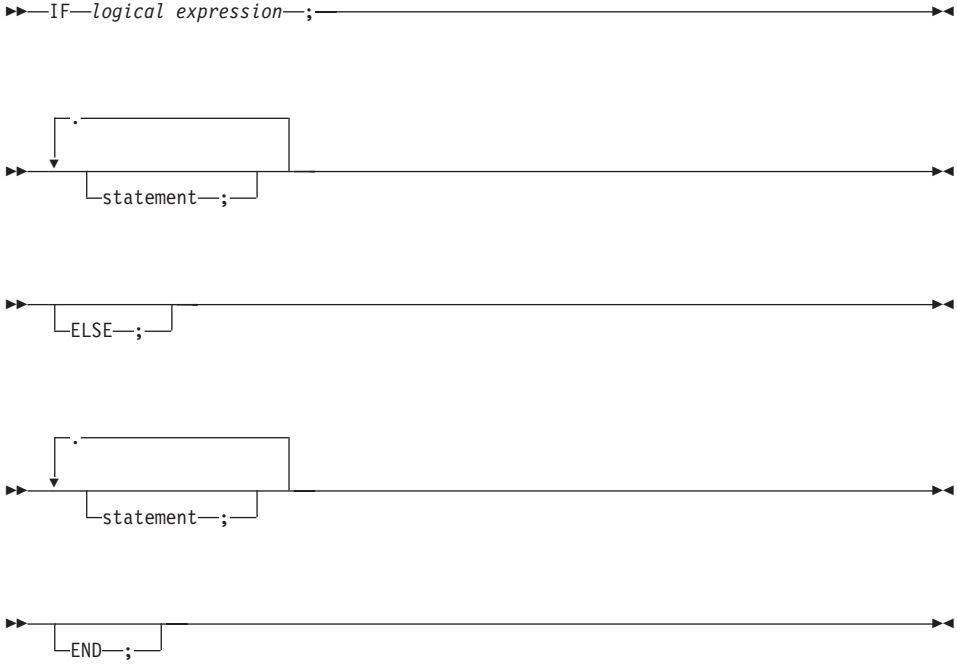
```
BIGGESTNUM = FUNCMAX(FIRST,SECOND);
```

To pass numeric literals as arguments and return the result to BIGGESTNUM:

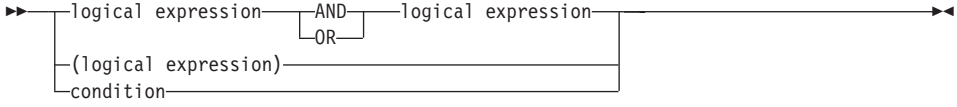
```
BIGGESTNUM = FUNCMAX(010,100);
```

IF statement

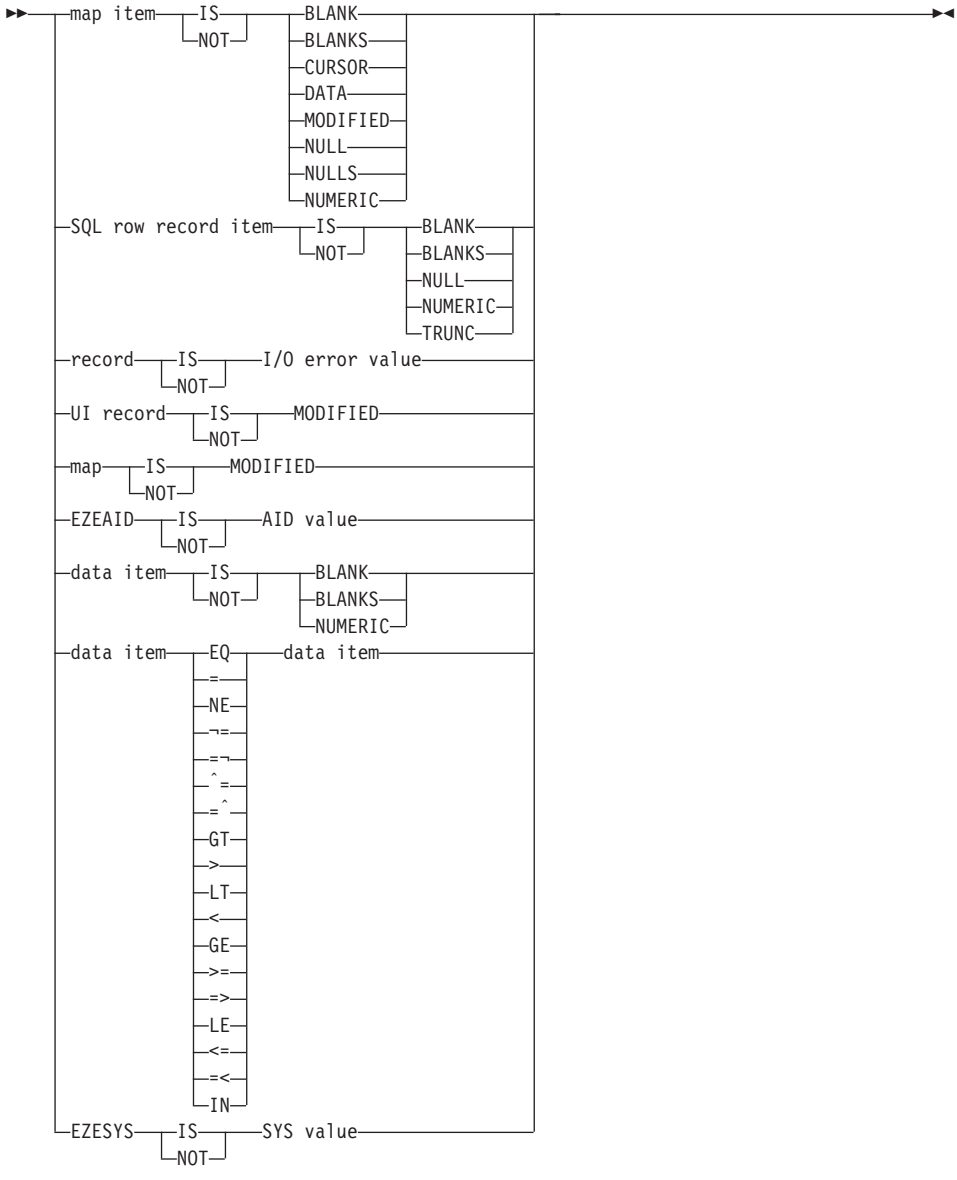
IF marks the start of a set of statements that is executed conditionally based on the results of one or more comparisons.



logical expression



condition



Attribute	Description
statement	Any statement or the line that represents the execution of the I/O option within statement definition.

Attribute	Description
AND, OR	Connectors that can be used to test multiple conditions. With AND, both conditions must be met. With OR, either condition can be met. A combination of AND and OR can be used, but AND is evaluated before OR unless you use parentheses to control the order. Multiple ANDs and ORs can be specified on a single line.
map item	Name of a variable field on a map or a map item parameter for a function. A map item can be subscripted, qualified, or both. This comparison is only valid for terminal maps.
IS	Boolean operator that tests true if the specified state is true.
NOT	Boolean operator that tests true if the specified state is false.
BLANK, BLANKS	When used with map items, tests true if either of the following cases are true: • The data received from the display for the specified data item contained all blanks or nulls or both. • The map containing the item has not been conversed since the program started, or since the last SET map CLEAR. When used with non-map items with data type CHA, MIX, DBCS, or UNICODE, tests true if the data item contains all blanks.
CURSOR	Tests that the user left the cursor in the specified data item.
DATA	Tests that there is data other than blanks or nulls within the map item specified. Either the user entered the data or the data was moved to the field before writing to the screen.
MODIFIED	Tests true if data in the variable field has changed. Data is considered changed if any of the following conditions are true: • When specified for a map variable field, data was entered by the program user the last time the map was displayed. • A SET MODIFIED was done prior to the CONVERSE of the map. • The field on the map was defined with a modified data tag (MDT) at map definition time, and this is the first display of the map in the program or the first display of the map after a SET CLEAR. • When specified for a map, tests true if any variable field on the map was changed. **Note:** This saves you from having to test each map field separately.

Attribute	Description
NULL, NULLS	When specified for map variable field, tests true if either of the following cases are true: • The data received from the display for the specified data item contained all nulls, blanks, or both. Nulls are received when the program user presses the Erase EOF key. • The map containing the item has not been conversed since the program started, or since the last SET map CLEAR. When used with non-map items with data type CHA, MIX, DBCS, or UNICODE, tests true if the data item contains all blanks.
NULL (SQL row record item)	Tests true if the SQL row record item has had no value assigned to the item.
NUMERIC	If the map item or data item type is character or mixed, tests true if the field contains the characters 0 through 9. NUMERIC cannot be used with EZE words.
SQL row record item	Name of a data item in an SQL row record or an SQL item parameter for a function. The name can include a qualifier.
TRUNC	Tests whether a character or a DBCS item in an SQL row record was truncated (nonblank characters deleted on the right) the last time the item value was read from the relational database. Truncation can only occur when the column in the database is longer than the data item. The TRUNC indicator is reset whenever a value is moved to the item, or when the item is set to NULL.
record	Name of a record.
I/O error value	Tests true if the I/O error value specified was returned from the system on the last I/O option that accessed the record. See “I/O error value” on page 368 for more information.
map	Name of a map.
EZEAID	The special function used to test the key that caused the input interrupt from the display.
AID value	Used in testing the state of the EZEAID special function word. See “AID value” on page 364 for more information.
data item	A data item syntactical element. See “Data item” on page 366 for more information.
EQ or =	Boolean operators that test true if data item values are equal.

Attribute	Description
NE, ¬=, =¬, ^=, or =^	Boolean operators that test true if data item values are not equal. **Note:** The ¬ and ^ symbols are not in the national language syntactic character set, and might not have an equivalent code point across different code pages. If you are exporting your program or generating for machines with differing code pages (in particular, between System/370 host systems and workstations), use NE, not the symbols.
GT or >	Boolean operators that test true if the value of the first data item is greater than the second.
LT or <	Boolean operators that test true if the value of the first data item is less than the second.
GE, >=, or =>	Boolean operators that test true if the value of the first data item is greater than or equal to the second.
LE, <=, or =<	Boolean operators that test true if the value of the first data item is less than or equal to the second.
IN	Boolean operator that tests true if the value in the first data item can be found in the array represented by the second data item. If a match is not found, processing skips to the corresponding ELSE or END statement. **Note:** The value of special function word EZETST is set to 0 if a match is not found. If a match is found, EZETST is set to the index number of the first element of the array that matches the value of the data item. Successive items in the array are compared until a match is found or the end of the array is reached. If the array includes an index, the testing starts there rather than from the first item in the array. If no starting index is given, the test starts with the first item in the array. If the value of the starting index is greater than the number of entries in the array or if no match is found, the test will test false. Comparing against a single data item instead of an array is equivalent to comparing for equal, but is slower and causes setting of EZETST to 0 or 1. It will not be treated as an error. The IN function is similar to the FIND statement in that they both scan for values, but you would use IF or WHILE rather than FIND in the following situations: • IN works with any array, not just a table column. • The search does not have to start at the first entry of the array. • Duplicate values can be found in the array.

Attribute	Description
EZESYS	A special function word used to test the system on which a program is running. The EZESYS test is a runtime test. Generation for a target system will fail if the program includes functions not supported on that system, even if the function is within an IF EZESYS clause that would prevent that function from executing on the target system. To allow generation for the target system to proceed, replace the offending function with a call to a program that performs the function.
SYS value	Used to test the state of the EZESYS special function word. See "SYS value" on page 377 for more information.

Definition considerations for IF

Statements between the IF logical expression and the ELSE statement (or END statement, if the ELSE is omitted) are processed only if the Boolean expression tests true. Any statements between the ELSE and the END are processed only if the Boolean expression tests false.

Parentheses can be used to control how conditions are evaluated.

When a conditional expression is nested within parentheses, evaluation proceeds from the least inclusive to the most inclusive part of the expression. The nested expression is evaluated before the expression which contains it. Unless the evaluation order is modified by parentheses, the AND operator is evaluated before the OR operator.

Parentheses can be used to:

- Modify the normal Boolean precedence of operations
- Eliminate ambiguities where operations appear at the same level.

The block of statements controlled by a conditional statement can contain conditional statements. This can continue up to a maximum of 15 levels deep.

When using the IF statement within a function, you may test the map attributes of a parameter item as long as the parameter item has been defined as a map item parameter or the SQL attributes as long as the parameter item has been defined as an SQL item parameter. This capability allows reusable routines to be written to handle the map and SQL item processing.

The following table shows which data items can be compared with each other:

Table 18. Valid data item comparisons

	BIN	CHA	DBCS	HEX	MIX	NUM	NUMC	PACK	PACF	UNICODE
BIN	x					x	x	x	x	
CHA		1		2	1	3				
DBCS			1							
HEX		2		4						
MIX		1			1					
NUM	x	3				x	x	x	x	
NUMC	x					x	x	x	x	
PACF	x					x	x	x	x	
PACK	x					x	x	x	x	
UNICODE										1

Legend:

x Valid data item comparison

1 For CHA to CHA, DBCS to DBCS, MIX to MIX, UNICODE to UNICODE, CHA to MIX, or MIX to CHA comparisons,the shorter item is logically padded on the right with blanks to the length of the longer item. All comparisons are logical comparisons.

2 Valid only if CHA field contains hexadecimal characters (a-f, A-F, 0-9). If a HEX item is compared to a CHA item, the CHA item is converted to HEX format, the shorter field is padded on the right with binary zeros, and a logical comparison is made.

3 Indicates that the data content of the source is validated prior to comparison. If nonnumeric, the program is abnormally terminated. Valid only if the numeric field is defined without decimal positions. The shorter field is padded on the left with zeros.

4 If a HEX item is compared to a HEX item, the shorter field is padded on the right with binary zeros to the length of the longer field, and a logical comparison is made.

Target environments for IF

ASCII character sets are used in workstation environments. EBCDIC character sets are used in host environments. Differences in collating sequence can cause greater-than or less-than comparisons to have different results in ASCII environments than in EBCDIC environments.

Environment	Compatibility Considerations
VM CMS	Uses EBCDIC character sets.
VM batch	Uses EBCDIC character sets.
CICS for MVS/ESA	Uses EBCDIC character sets.
MVS/TSO	Uses EBCDIC character sets.
MVS batch	Uses EBCDIC character sets.

Environment	Compatibility Considerations
IMS/VS	For a map field to test true when the data entered for a data item contained all blanks, nulls, or a combination of both, the program user must enter at least one blank in the field before pressing the Erase EOF key. If the program user presses Erase EOF without entering one blank in the field, IMS message format services leave the field set to its original contents. Uses EBCDIC character sets.
IMS BMP	Uses EBCDIC character sets.
CICS for VSE/ESA	Uses EBCDIC character sets.
VSE batch	Uses EBCDIC character sets.
CICS for OS/2	Range check comparisons for character data are performed using the ASCII collating sequence.
OS/400	Uses EBCDIC character sets.
OS/2 (GUI)	Range check comparisons for character data are performed using the ASCII collating sequence. The following are not supported: • IF record IS I/O error value • IF record NOT I/O value
Windows (GUI)	Same as OS/2 (GUI)
OS/2 (C++)	Uses ASCII character sets.
AIX	Uses ASCII character sets.
HP-UX	Uses ASCII character sets.
CICS for AIX	Uses ASCII character sets.
Windows NT (C++)	Uses ASCII character sets.
Windows NT (Java)	Uses ASCII character sets.
CICS for Windows NT	Uses ASCII character sets.
Solaris	Uses ASCII character sets.
CICS for Solaris	Uses ASCII character sets.
Test Facility	No distinction is made between testing for BLANKS and NULLS. Uses ASCII character sets.

Examples for IF

The following are examples of the IF statement:

IF ELSE statement

```
IF FLD1 EQ FLDA
  AND FLD2 EQ FLDB;
   PROCDUP();
ELSE;
   GETMOR();
END;
```

Nested IF statements

To use nested IF statements to respond to the function keys pressed by a user, type:

```
IF EZEAID IS PF3
  OR EZEAID IS PA2;
  EZECLOS;
ELSE;
  IF EZEAID IS PF12;
     PROCINQ();
  ELSE;
      IF EZEAID IS PF10;
          PROCADD();
      END;
  END;
END;
```

IF statement with AND and OR conditions

The following is an example of an IF statement using parentheses and mixing AND and OR conditions:

```
 IF NUMRECS > 0 AND                       /* Ensure there are records
 (OPCODE = CHGREC OR OPCODE = REVREC OR   /* to change, review,
  OPCODE = DELREC);                       /* or delete
```

IF statement testing map data

The following is an example of an IF statement that tests if a map contains data:

```
 IF DEPTNO IS DATA;          /* If user has entered data in the dept
  LOOKUP();                   /* field, find department information.
```

IF statement comparing numeric and character data

The following example demonstrates how the comparison of numeric to character data would work:

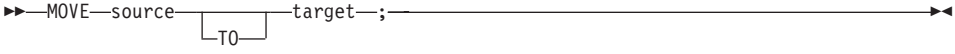
```
ITEM-A is defined as CHA, length 4
ITEM-B is defined as NUM, length 4

MOVE '3' TO ITEM-A;         /* ITEM-A's value is '3'
MOVE 3 TO ITEM-B;           /* ITEM-B's value is '0003'
IF ITEM-A EQ ITEM-B;
   EQ-FUNCTION();           /* This will never be done
ELSE;
   NE-FUNCTION();           /* This will always be done
END;
MOVE '0003' TO ITEM-A;      /* ITEM-A's value is now '0003'
```

```
IF ITEM-A EQ ITEM-B;
  EQ-FUNCTION2();           /* This will always be done
ELSE;
  NE-FUNCTION2();           /* This will never be done
```

MOVE statement

The MOVE statement moves the contents of one item to another item, or moves the corresponding items in one data structure to another data structure.

Any statement that can be written as a MOVE statement can also be written as an assignment statement.



Attribute	Description
source	A literal, data item (can be subscripted, qualified, or both), record, map, or certain special function words. A literal is limited to the size of the target.
target	A data item (can be subscripted, qualified, or both), record, map, or certain special function words. If the source is a literal, data item, or special function word, the target must be a data item or special function word. If the data item has an OCCURS greater than 1 and no subscript is supplied, the first occurrence of the data item is used. The data item might be defined in a record, map, or table or in the function parameter or local storage lists or in one of the records in these lists. If the source is a record or map, the target must be a record or map. The data items within a record or map move to the corresponding data items, with the same name, within the record or map that is the target.

Definition considerations for MOVE

The following table shows the valid source and target data item types:

Source	Target
BIN	BIN, NUM, NUMC, PACF, PACK
CHA	CHA, HEX [5], MIX, NUM [6]
DBCS	DBCS
HEX	CHA, HEX
MIX	CHA, MIX

NUM	BIN, CHA [7], NUM, NUMC, PACF, PACK
NUMC	BIN, NUM, NUMC, PACF, PACK
PACF	BIN, NUM, NUMC, PACF, PACK
PACK	BIN, NUM, NUMC, PACF, PACK
UNICODE	UNICODE

Moved Data Exceptions

Generally, the exact data content is moved from a source data item to the target data item. There are five exceptions to this:

1. A MOVE between NUM, NUMC, PACK, PACF, and BIN data items results in the necessary format conversions being made.
2. A MOVE statement between data items with unequal lengths results in truncation or padding depending on the data type.

   If the target is a CHA, DBCS, or UNICODE item, the source value is truncated or padded on the right with blanks as required.

   If target is a HEX item, the move takes place left to right, truncating or padding on the right with binary zero bytes as required.

   If the target item is numeric, packed, or binary, the source data is first decimally aligned to match the number of decimal places in the target. The source is then moved to the target with excess digits on either side of the decimal point truncated. If there are fewer digits on either side of the decimal point, zeros are added.

   If a MIX data item is moved to a longer data item, the target is padded on the right with single-byte blank characters. If the target item length is shorter than the source MIX data item length, the source data must be truncated. Unoccupied positions in the target that result from DBCS substring truncation are filled with single-byte blank characters.
3. In a move from HEX to CHA data, the HEX field is converted to hexadecimal character representation (0-9, a-f, A-F). Each HEX byte is converted to two character bytes. The move is done left to right, truncating or padding with character zeros as required.
4. In a move from CHA to HEX data, the character field must contain only the characters a-f, A-F, or 0-9. Each pair of characters from the character field is translated to its single HEX byte equivalent. The move is done left to right, truncating or padding with binary zeros as required. Execution is

5. Valid only if the CHA field contains hexadecimal characters (a-f, A-F,0-9)

6. Indicates that the data content of the source is validated prior to movement. If the data content is nonnumeric, the program is abnormally terminated. This movement is valid only if the numeric field is defined without decimal positions.

7. This movement is valid only if the numeric field is defined without decimal positions.

terminated if the CHA field contains characters that are not valid for HEX conversion. You can use the hexadecimal variable field edit to ensure that data entered from a map is valid for HEX conversion.

5. Moving from NUM data items to CHA data items does no conversion (this can only be done if the numeric field contains no decimal positions). In other words, the numeric field is treated as if it were character. If you wish to move the NUMC data format so that the sign for positive numbers is converted, the low-order byte that contains the sign can be converted as follows:
   - Move a NUMC item to a NUM item.
   - Move the NUM item to a CHA item.

   If the NUM or NUMC item has a negative value, the last byte is an invalid character.

Move Corresponding

Data moved between two structures with a single statement is called a move corresponding. These structures can be records or maps. Level-77 items are not considered part of a record structure and are not included in the move. The generated program operates as if one MOVE statement was specified for each item (or map variable field) in the source structure that has an item (or map variable) with the same name in the target structure.

Move corresponding is useful when moving data between maps and records that have corresponding map fields and data items. When moving entire records it is better to use a MOVE between the two high-level data items of the records rather than doing a move corresponding. Both accomplish the same thing, but the high-level data item MOVE executes one MOVE instead of a MOVE for each data item. If a high-level data item is used, be sure that the data items defined in both structures match in length and type because no data conversion will be done.

Similarly, if you are moving part of your record to another record, it is more efficient to move the highest level structures possible in the records.

When moving data from a record or table to a map, you should be sure the record data can be displayed. If a character data item in a record contains data that cannot be displayed, it might cause terminal errors to occur when moved to a map. If a field exists in both the record and the map (the field has the same name) and it is binary or packed in the record, it must be numeric in the map.

Target environments for MOVE

Supported in all environments without compatibility considerations.

Examples for MOVE

Following are examples of the MOVE statement:

MOVE statement

```
MOVE STATE TO DSTAT;
```

Moving a Blank to a Data item

The following moves a blank to a data item called ITEM1:

```
MOVE ' ' TO ITEM1;
```

VisualAge Generator does not support keywords for the MOVE statement as are supported by COBOL, such as BLANK, BLANKS, ZERO or ZEROS. Use literals instead.

Only one literal blank is needed regardless of the field length.

Moving Zero to a Numeric or Binary Field

The following fills a numeric or binary field called ITEM2 with zeros:

```
MOVE 0 TO ITEM2;
```

Only one 0 is needed regardless of the field length.

Moving Fields from One Map to Another

The following moves all fields with identical names from MAP1 to MAP2:

```
MOVE MAP1 TO MAP2;
```

Moving a Data item to an Element of an Array

The following moves the contents of ITEM1 to the second occurrence of ARRAY in REC1:

```
MOVE ITEM1 TO REC1.ARRAY[2];
```

Using special function words in a MOVE statement

Some special function words can be used in a MOVE statement.

The following moves a program user ID to an item called NAME:

```
MOVE EZEUSRID TO NAME;
```

The following moves a terminal ID to an item called TERM-ID:

```
MOVE EZELTERM TO TERM-ID;
```

The following moves the current date to an item called DATE:

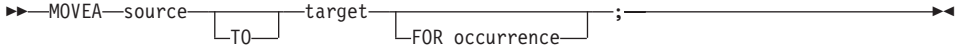
```
MOVE EZEDTE TO DATE;
```

The following moves a literal to a map message field (EZEMSG):

```
MOVE "This is a message" TO EZEMSG;
```

MOVEA statement

The MOVEA statement moves the contents of one array to another or initializes the elements of an array.



Attribute	Description
source	A literal, data item (can be subscripted, qualified, or both), array, or special function word. The source data type must be compatible with the target data type.
target	An array or a data item in an array or table.
occurrence	Any numeric literal, data item (can include a subscript, be qualified, or both), or EZE special function word that contains an integer greater than 0 but less than 65536.

Uses

The MOVEA statement simplifies the coding of a program by replacing a loop or a series of MOVE statements for the following actions:

- Initializing arrays
- Moving tables or parts of tables into map or record arrays
- Moving large arrays into small map arrays and using the CONVERSE statement to present them to the program user.

If the source is an array (map variable field array, table column, or item in a record with multiple occurrences), MOVEA moves the source array to the target array. If the source is a literal value or scalar (single-valued item or field), the target array is initialized with the scalar. You can designate a starting position within each array and specify the number of elements you want to move.

Subscripts

The source and target can each include a subscript. The subscript specifies the starting position within that array for the move array operation. If you do not use a subscript, MOVEA starts with the first element of the array. The number of occurrences from the starting position specified to the end of the array is called the resultant size. For example, if the array has 10 elements and 3 is specified as the subscript, the size is 8.

EZETST contains the subscript of the last element changed in the target.

Definition considerations for MOVEA

For definition considerations for MOVEA, see "Definition considerations for MOVE" on page 411.

Target environments for MOVEA

Supported in all environments without compatibility considerations.

Examples for MOVEA

Following are examples of the MOVEA statement.

Scalar to array with MOVEA

The source is a literal or a scalar item (not an array). The FOR operand specifies the number of elements to which the source value is propagated. If the FOR operand is omitted, the default is the resultant size of the target array.

```
MOVEA source TO target[x] FOR y;
```

If y is less than or equal to the resultant size of the target, this statement moves the value of the source to elements x through (x + y - 1). However, if y is larger than the resultant size of the target, or is omitted, the MOVEA statement moves the value of the source to elements x through the end of the array. This function of the MOVEA statement is ideal for initializing arrays.

Array to array with MOVEA

The source must be an array element. The FOR operand specifies the number of items to be moved. If the FOR operand is omitted, the default is the smaller resultant size of either the source or the target.

```
MOVEA source[x] TO target[y] FOR z;
```

This statement moves the contents of the source, beginning with element x, to the target, beginning with element y, for the minimum of the resultant size of the source, the resultant size of the target, and z.

For instance, suppose the source had 5 elements, the target had 10, and the following statement was used:

```
MOVEA source TO target [5] FOR 3;
```

Only three elements are moved since that is the minimum of 5, 6, and 3. The first three elements of the source are moved, by position, to the fifth, sixth, and seventh elements of the target.

Initializing an entire array with MOVEA

```
MOVEA 0 to ARRAY2;

BEFORE   source = 0    ARRAY2 =  1  2  3  4
AFTER    source = 0    ARRAY2 =  0  0  0  0

EZETST = 4
```

Because an entire array was initialized, EZETST holds the size of the array.

Initializing part of an array with MOVEA

```
MOVEA 'A' TO ARRAY2[2];

BEFORE   source = 'A'    ARRAY2 =  1  2  3  4
AFTER    source = 'A'    ARRAY2 =  1  A  A  A

EZETST = 4
```

Because an occurrence was omitted, the default is the resultant size (4 - 2 + 1 = 3) of the target array. EZETST holds the subscript of the last element changed in the target array.

Character string to array with MOVEA

```
MOVEA 'ABC' TO ARRAY2;

BEFORE   source = 'ABC'    ARRAY2 =  1     2     3    4
AFTER    source = 'ABC'    ARRAY2 =  ABC  ABC  ABC  ABC

EZETST = 4
```

A character string has been moved into each element of the array.

Data item to array with MOVEA

```
MOVEA RESULT TO ARRAY2[2] FOR 3;

BEFORE   RESULT = 'ABC'    ARRAY2 =  1   2     3    4
AFTER    RESULT = 'ABC'    ARRAY2 =  1   ABC  ABC  ABC

EZETST = 4
```

By using the subscript and FOR options, source was moved to target beginning with the second element in target and ending with the fourth element.

Changing part of an array with MOVEA

```
MOVEA ARRAY1[1] TO ARRAY2[2] FOR 2;

BEFORE   ARRAY1 =  A  B  C     ARRAY2 =  1  2  3  4
AFTER    ARRAY1 =  A  B  C     ARRAY2 =  1  A  B  4

EZETST = 3
```

By using the subscript and FOR options, only part of an array was changed. Notice that the first and last elements in ARRAY2 did not change.

Target array smaller than source array with MOVEA

```
MOVEA ARRAY1 TO ARRAY2;

BEFORE   ARRAY1 =  A  B  C     ARRAY2 =  1  2
AFTER    ARRAY1 =  A  B  C     ARRAY2 =  A  B

EZETST = 2
```

Notice that the third element in ARRAY1 did not move.

Target array larger than source array with MOVEA

```
MOVEA ARRAY1 TO ARRAY2;

BEFORE   ARRAY1 =  A  B  C     ARRAY2 =  1  2  3  4
AFTER    ARRAY1 =  A  B  C     ARRAY2 =  A  B  C  4

EZETST = 3
```

Notice that the fourth element in ARRAY2 did not change.

Move array in record or table to map array with MOVEA

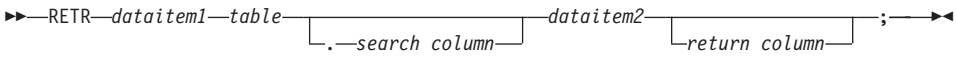
```
MOVEA ARRAY1[START] TO ARRAY2;
```

In this example, the MOVEA statement is used to move an array in a data structure to a map array. ARRAY1 has 100 elements and ARRAY2, which is on a map, has only 10.

The variable, START, can then be modified to step through the array. If the map is conversed 10 times to display all the information in ARRAY1, START could be set to 1 on the first converse, 11 on the second converse, 21 on the third, and so forth. The above statement moves the data in sets.

RETR statement (Retrieve)

RETR (Retrieve) obtains data from a table based on a search argument.



Attribute	Description
dataitem1	Data item name or literal. The data item can be subscripted, qualified, or both.
table	Name of a table.
search column	Name of a column in the table. The default is the first column in the table.
dataitem2	Data item name (cannot be literal). The data item can be subscripted, qualified, or both. This data item receives the value of the return column.
return column	Name of a column in the table. The default is the second column in the table.

Definition considerations for RETR

If the value in `dataitem1` is found in the search column of a row in the table, the data in the return column of the same row is moved to `dataitem2`. If the data in the search column is not unique, the first occurrence in the table is the one used.

The special function word EZETST is loaded with a value depending on the results of the RETR statement. The contents of EZETST will be either:

- Zero if the data item is not found
- The row number where the data item is located if the data item is found.

When EZETST contains a row number, it can be used as a subscript for other statements that reference other columns in that same row of the table.

Target environments for RETR

Supported in all environments without compatibility considerations.

Examples for RETR

In the following example, a table (INFO) has 50 rows and 3 columns called STATE, POPULATION, and AREA. The first column has an entry for each of the 50 states, the second column contains the population for each state, and the third column contains the area in square miles for each state.

INFO:

STATE	POPULATION	AREA
Alabama	3,500,000	51,600

Alaska	302,000	586,000
.	.	.
.	.	.

A RETR statement could be used to pick up the area information from the above table, based on a matching state.

```
MOVE 'ALASKA' TO ITEM;
RETR ITEM INFO.STATE AMOUNT AREA;
```

AMOUNT now has 586,000 in it. EZETST contains 2, the row number of the matching state. If no match is made, EZETST is set to 0 and the contents of AMOUNT are not changed. If the match is found, you can now obtain the population for ALASKA by the following statement:

```
MOVE INFO.POPULATION[EZETST] TO PEOPLE;
```

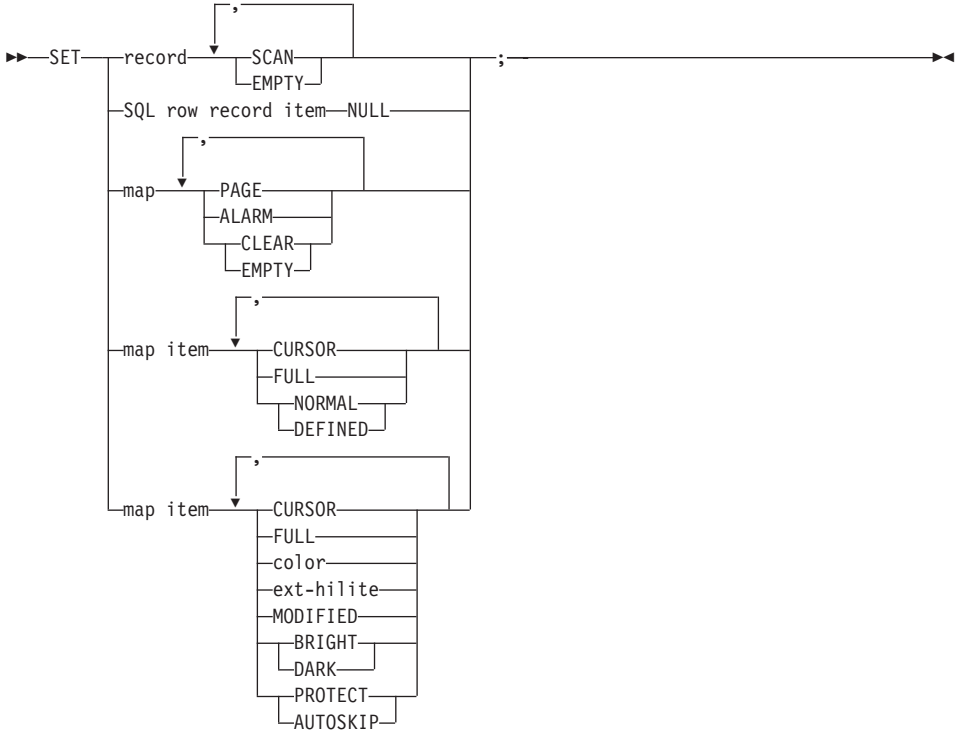
You could also code:

```
PEOPLE = INFO.POPULATION[EZETST];
```

SET statement

You can use SET to do any of the following:

- Position the cursor on a map field
- Change the attribute of a map field
- Set an SQL row record item to null
- Clear a map or record
- Skip to a new page
- Establish a position in a file



color

ext-hilite (extended highlighting)

Attribute	Description
record	Name of a record.
SQL row record item	Name of a data item in an SQL row record or an SQL item parameter for a function. The item name can be qualified by the record name.
map	Name of a map.
map item	Name of a variable field on a map or a map item parameter for a function. A map item can be subscripted, qualified, or both.
SCAN	Used to establish the scan position for an indexed file or DL/I database without having to use an INQUIRY I/O option. If the record is an indexed file, the action taken depends on the next option selected. • If the option executed for the file is a SCAN, the next record retrieved from a file will have a key value greater than or equal to the record ID item defined for the record. • If the option executed for the file is a SCANBACK, the record retrieved will be the record with the highest key value that is less than or equal to the record ID item defined for the record. • Options other than SCAN and SCANBACK cause the condition to be reset and ignored. A SET record SCAN with a key value set to all X'FF' bytes prior to a SCANBACK sets the position pointer in all environments to the end of the file, so that the next SCANBACK retrieves the last record in the file. For DL/I records, you must set the segment key value and the key values of its parent segments, if there are parent segments, in the DL/I database. If the record is in a DL/I database and the next option executed for the record is a SCAN using the default search arguments, then the DL/I call is modified to retrieve the first occurrence of the record in the database at or following the position indicated by the segment key value and the key values of its parent segments in the database. SET record SCAN can be used only with DL/I calls with Scan in Parent equal to No and unmodified SSA lists. If the next option executed for that segment is a SCAN DL/I function with a modified SSA list, execution is terminated. If the next option executed for the segment is not a SCAN, the condition is reset and ignored.

Attribute	Description
EMPTY	Used to initialize all data items in a record (blanks for character, mixed, DBCS and Unicode data items; zeros for numeric data items; and binary zeros for binary and hexadecimal data items). If the record is a working storage record, the level-77 items are not affected by the SET record EMPTY statement. For a data structure that is subdivided, a SET record EMPTY statement does the same thing as specifying individual MOVE statements of the default values for each data item in the structure. If specified for a map, EMPTY causes the contents of each map field to be set to 0 for numeric fields, or blanks for character, mixed, and DBCS fields. The field attributes are not changed.
NULL	Sets the null indicator for an item in an SQL row record structure to the null condition (value -1). The condition has no effect on items not in SQL row records.
PAGE	Used to clear the display or advance the paper to the top of the next page before the next CONVERSE or DISPLAY I/O option. This condition affects the device for the next CONVERSE or DISPLAY I/O option for any map, not just the one specified in the SET statement. An automatic SET PAGE is performed when a different fixed map appears that would be positioned on any of the same lines as defined in an already displayed fixed map (unless the maps match exactly on start position and depth). If this map is the first in a program and is going to a printer, the program user must position the paper to the top of the page before running the program, unless a SET map PAGE is used. If a called and calling program are both printing to the same file, a CLOSE map can be issued before the first DISPLAY in the called program to ensure that the paper is positioned correctly for the called program.
ALARM	Causes the display alarm to sound on the next CONVERSE I/O option when the specified map appears.
CLEAR	Resets the map to its originally defined state. The specified map does not have to be the next map displayed. The attributes and contents for the fields are set to the original values defined for the map.
CURSOR	Positions the cursor on the first position of a map field when the map appears. If more than one SET map field CURSOR is issued for a map, the cursor is positioned on the map field of the last SET statement issued. The CURSOR state does not affect any other condition.

Attribute	Description
NORMAL	Displays data with normal intensity, and a map field is set to UNPROTECTED and UNMODIFIED. The only conditions that can be used with NORMAL are CURSOR and FULL.
DEFINED	Displays data with the attributes originally defined for the map field. The only conditions that can be used with DEFINED are CURSOR and FULL.
FULL	Used to remind the program's users that a field should be completely filled. Asterisks are placed in an empty or blank field when the map is displayed by the CONVERSE I/O option. When the program user types data in the variable field, any remaining asterisks are part of the data. If a map field has a fill character other than the default specified, the fill character is used instead of the asterisks. The asterisks only appear on the map, and not in internal storage if the map field is not set to MODIFIED. The program user must clear the field of the asterisks if data is entered in the fields to prevent the asterisks from being passed to the program. FULL remains in effect until another SET map item statement for this field executes or something else is done that clears the field attributes, such as a SET map CLEAR. SET FULL is for each individual item, not the map structure. SET of an empty mixed map field FULL causes single-byte asterisks to appear in the map field the next time it is displayed. To be considered empty, the map field must contain all single-byte blanks. If the program wants the use of SET map item FULL to be honored, the map group must be generated with the /SETFULL generation option. If the /NOSETFULL generation option is specified, the mapping services program will not place an asterisk (*) in fields that have been set full.
color	Sets the color attribute for a map field to one of the following colors: monochrome, blue, pink, yellow, turquoise, red, green, or white.
ext-hilite	Sets the extended highlighting attribute for a map field to one of the Following values: no extended highlighting, blink, reverse video, or underscore.

Attribute	Description
MODIFIED	Used to set the status of the variable field to modified. This forces the contents of the field to be returned to the program on the next CONVERSE. The modified condition does not have any affect on the map field until the map is conversed. A TEST MODIFIED statement before a map is conversed does not give a true condition as a result of this SET statement. This statement can also be used to force editing of the data (non-blank character or non-zero numeric) in a field when the map is conversed. The modified condition is reset before the next display of a map unless another SET map field MODIFIED statement is executed.
BRIGHT	Displays a map field with bright intensity. If you are running on a color display and all your map fields are defined with default colors, the BRIGHT operand causes the color of a normal unprotected field to be changed from green to red. (A normal protected field changes from blue to white.) Otherwise, no color change takes place.
DARK	Used to prevent the variable field from displaying data. DARK is usually used for security reasons, such as passwords.
PROTECT	Protects a map field from modification by a user. PROTECT and AUTOSKIP cannot be specified together because AUTOSKIP forces PROTECT.
AUTOSKIP	Protects a map field from modification by a user, by causing the cursor to skip over this field.

Definition considerations for SET

When using the SET statement to set a map field with attributes other than cursor, color, or extended highlighting, the specified attributes act as a complete replacement for all attributes other than cursor, color, extended highlighting, numeric, and fold attributes specified at map definition.

When using the SET statement within a function, you may set the map attributes of a parameter item as long as the parameter item has been defined as a map item parameter or the SQL attributes as long as the parameter item has been defined as an SQL item parameter. This capability allows reusable routines to be written to handle the map and SQL item processing.

When using the SET statement to assign color and extended highlighting attributes to variable map items, you can specify only one color and one extended highlighting attribute. They can be combined with any other map

item attribute values except DEFINED and NORMAL. Both DEFINED and CLEAR reset the color and extended highlighting attributes to those originally defined in map definition. NORMAL has no effect on either color or extended highlighting.

The following guidelines also apply to the color and extended highlighting attributes:

- If you issue more than one SET statement containing color or extended highlighting for a map item before the map is displayed, the selection that was specified last is used.
- If you are using a color display and assign a color to a field on a map containing all monochrome fields, execution will switch from four-color mode to seven-color mode. To prevent colors from changing inadvertently, you can explicitly assign colors to the fields on the map in map or program definition instead of accepting the monochrome default.
- If you specify a combination of regular highlighting (BRIGHT, DARK), color, or extended highlighting attributes for a map item, the resulting appearance of the item is device-dependent. Some of the attributes you set might not be visible on the map.

Target environments for SET

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
OS/2 (GUI)	Only statements supported are: • SET record EMPTY • SET SQL row record item NULL
Windows (GUI)	Same as OS/2 (GUI).
OS/2 (C++)	None.

Environment	Compatibility considerations
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	None.

Examples for SET

The following resets MAP1 to its original state:

```
SET MAP1 CLEAR;
```

The following clears the panel and resets MAP1 to its original state before it appears, and to sounds the terminal alarm when MAP1 appears:

```
SET MAP1 PAGE, CLEAR, ALARM;
```

The following sets all the fields in a record called REC1 to zero (numeric data items) or blank (character data items):

```
SET REC1 EMPTY;
```

The following sets the color to red and the extended highlighting to blink:

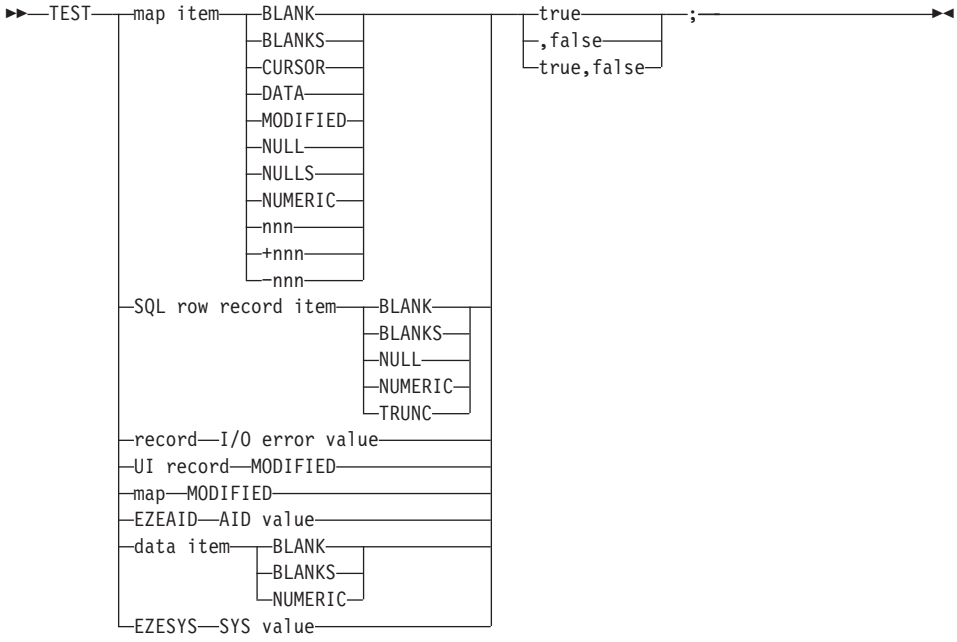
```
SET MAPITEM RED,BLINK;
```

The following sets the extended highlighting to reverse video, to protect a map field, and sets the color to turquoise:

```
SET MAPITEM RVIDEO,PROTECT,TURQ;
```

TEST statement

TEST transfers control to a function based on the status of the tested field, record, map, or special function word with the EZE prefix.



Attribute	Description
map item	Name of a variable field on a map or a map item parameter for a function. A map item can be subscripted, qualified, or both. This comparison is only valid for terminal maps.
BLANK, BLANKS	When used with map items, tests true if either of the following cases are true: • The data received from the display for the specified data item contained all blanks or nulls or both. • The map containing the item has not been conversed since the program started, or since the last SET map CLEAR. When use with non-map items with data type CHA, MIX, or DBCS, tests true if the data item contains all blanks.
CURSOR	Tests that the user left the cursor in the specified data item.
DATA	Tests that there is data other than blanks or nulls within the map item specified. Either the user entered the data or the data was moved to the field before writing to the screen.

Attribute	Description
MODIFIED	Tests true if data in the variable field has changed. Data is considered changed if any of the following conditions are true: • When specified for a map variable field, data was entered by the program user the last time the map was displayed. • A SET MODIFIED was done prior to the CONVERSE of the map. • The field on the map was defined with a modified data tag (MDT) at map definition time, and this is the first display of the map in the program or the first display of the map after a SET CLEAR. • When specified for a map, tests true if any variable field on the map was changed. **Note:** This saves you from having to test each map field separately.
NULL, NULLS	When specified for map variable field, tests true if either of the following cases are true: • The data entered into the panel for the specified data item contained all nulls or blanks. Nulls are received when the program user presses the Erase EOF key. Note that a true TEST for NULLS does not mean that the field contains nulls internally (it contains blanks). • The map containing the item has not been conversed since the program started, or since the last SET map CLEAR. When use with non-map items with data type CHA, MIX, DBCS, or UNICODE, tests true if the data item contains all blanks.
NULL (SQL row record item)	Tests true if the SQL row record item has had no value assigned to the item.
NUMERIC	If the map item or data item type is character or mixed, tests true if the field contains the characters 0 through 9. NUMERIC cannot be used with EZE words.
nnn, +nnn, −nnn	Compares the length of the data returned against the value nnn, which is a numeric literal. If no sign precedes the number, the test is for an IS EQUAL TO condition. If a + sign precedes the number, the condition tested is IS GREATER THAN. For a minus (−) sign preceding a number, the test is IS LESS THAN. • In calculating the length, leading blanks, trailing blanks, and nulls are not counted. • If the field is at its originally defined state, the length is 0. For example, if the map item contains default text (inserted during definition) and it has not been modified during execution in any way, then the length is calculated as 0. SET map CLEAR resets a field to its originally defined state. • If the field is not at its originally defined state, then the length is calculated based on what was displayed or entered on the last converse.

Attribute	Description
true	Name of a main function or EZECLOS if specified within a program flow. The name of a function, EZEFLO, EZERTN, or EZECLOS if specified within a function.
false	Name of a main function or EZECLOS if specified within a program flow. The name of a function, EZEFLO, EZERTN, or EZECLOS if specified within a function.
SQL row record item	Name of a data item in an SQL row record or an SQL item parameter for a function. The name can include a qualifier.
TRUNC	Tests whether a character or a DBCS item in an SQL row record was truncated (nonblank characters deleted on the right) the last time the item value was read from the relational database. Truncation can only occur when the column in the database is longer than the data item. The TRUNC indicator is reset whenever a value is moved to the item, or when the item is set to NULL.
record	Name of a record.
I/O error value	Tests true if the I/O error value specified was returned from the system on the last I/O option that accessed the record. See “I/O error value” on page 368 for more information.
map	Name of a map.
EZEAID	The special function used to test the key that caused the input interrupt from the display.
AID value	Used in testing the state of the EZEAID special function word. See “AID value” on page 364 for more information.
data item	A data item syntactical element. See “Data item” on page 366 for more information.
EZESYS	A special function word used to test the system on which a program is running. The EZESYS test is a runtime test. Generation for a target system will fail if the program includes functions not supported on that system, even if the function is within an IF EZESYS clause that would prevent that function from executing on the target system. To allow generation for the target system to proceed, replace the offending function with a call to a program that performs the function.
SYS value	Used to test the state of the EZESYS special function word. See “SYS value” on page 377 for more information.

Definition considerations for TEST

If you specify both a true and a false name on a TEST statement, you can separate the names with a comma. If you specify only the false part of the

statement, you must precede it with a comma. Commas are automatically inserted for you when you use the TEST statement template.

If the state being tested is true, the name specified as the true attribute is executed as an unconditional statement. If the true attribute is not specified, execution continues with the statement following the TEST statement.

If the state being tested is false, the name specified as the false attribute is executed as an unconditional statement. If the false attribute is not specified, execution continues with the statement following the TEST statement.

Testing a map item is valid from the time the map is conversed until the next map appears or is conversed. If the modified data tag is on, the value of the item as it appears on the display is tested. Test statement results are consistent across environments for map items with fill characters null or blank, or with the modified data tag on.

When using the TEST statement within a function, you may test the map attributes of a parameter item as long as the parameter item has been defined as a map item parameter or the SQL attributes as long as the parameter item has been defined as an SQL item parameter. This capability allows reusable routines to be written to handle the map and SQL item processing.

Target environments for TEST

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	For a map field to test true when the data entered for a data item contained all blanks, nulls, or a combination of both, the program user must enter at least one blank in the field before pressing the Erase EOF key. If the program user presses Erase EOF without entering one blank in the field, IMS message format services leave the field set to its original contents.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.

Environment	Compatibility considerations
OS/400	None.
OS/2 (GUI)	The following are not supported: • TEST record IS I/O error value • TEST record NOT I/O error value
Windows (GUI)	Same as OS/2 (GUI).
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	None.
CICS for Windows NT	None.
Solaris	None.
CICS for Solaris	None.
Test Facility	No distinction is made between testing for BLANKS and NULL.

Examples for TEST

The following examples show how you can use the TEST statement.

TEST statement using NULL

In the following example, control transfers to AOK if the STA column in the SQLR record is null. Otherwise, control transfers to TRYGEN.

```
TEST SQLR.STA NULLS AOK,TRYGEN;
```

TEST statement using NUMERIC

To test using NUMERIC, type:

```
TEST TEMPNUM NUMERIC IS-NUM,NOT-NUM;  /* Call appropriate routine.
```

TEST statement using MODIFIED

To pass control to MOD if an item is modified, and pass control to NOMOD if it is not, type:

```
TEST ITEM MODIFIED MOD,NOMOD;
```

To pass control to MOD only if an item is modified, type:

```
TEST ITEM MODIFIED MOD;
```

If the item is not modified, the statement immediately following the TEST statement is executed.

To pass control to NOMOD if an item is not modified, type:

```
TEST ITEM MODIFIED ,NOMOD;
```

If the item is modified, the statement immediately following the TEST statement is executed.

Testing for a function key

To test if PF1 was pressed by the program user, type:

```
TEST EZEAID PF1 TRUEGP,FALSEGP;
```

A variation that is a test for a true condition only is:

```
TEST EZEAID PF1 TRUEGP;
```

The TRUEGP set of statements is executed only if (in this example) PF1 is pressed.

Use of test for false testing is as follows:

```
TEST EZEAID PF1 ,FALSEGP;
```

Testing for bypass edit PF keys or a PA key

To test for bypass edit PF keys or a PA key code the following:

```
TEST EZEAID BYPASS EZEFLO,EZERTN;
```

Testing the results of the last I/O for a record

You can use a TEST statement to test the results of the last I/O operation for a record.

The following statement runs ERR1 if REC1 has an end of file condition:

```
TEST REC1 EOF ERR1;
```

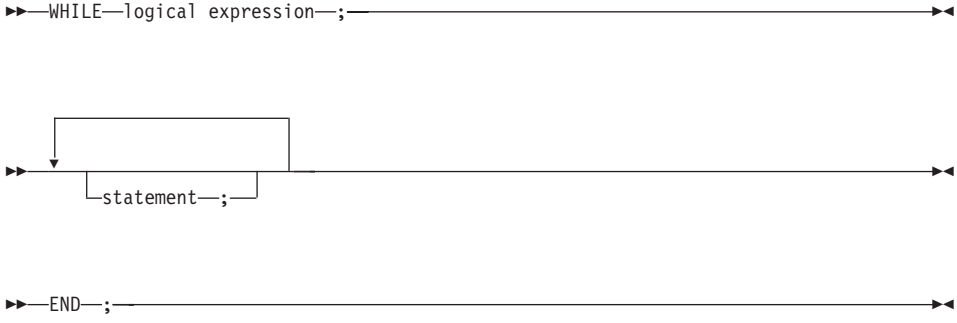
Testing for the length of data

If a field on map (MAP1.FLD1) is defined as length 10 and XYZ is entered, the length of the data entered is 3:

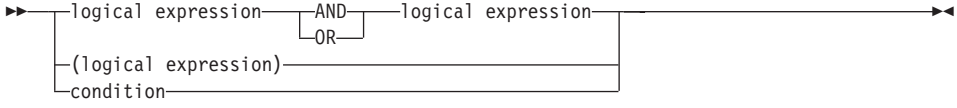
```
TEST MAP1.FLD1 +2 TRUEGP,FALSEGP;   /* true
TEST MAP1.FLD1 3 TRUEGP,FALSEGP;    /* true
TEST MAP1.FLD1 -3 TRUEGP,FALSEGP;   /* false
TEST MAP1.FLD1 +3 TRUEGP,FALSEGP;   /* false
```

WHILE statement

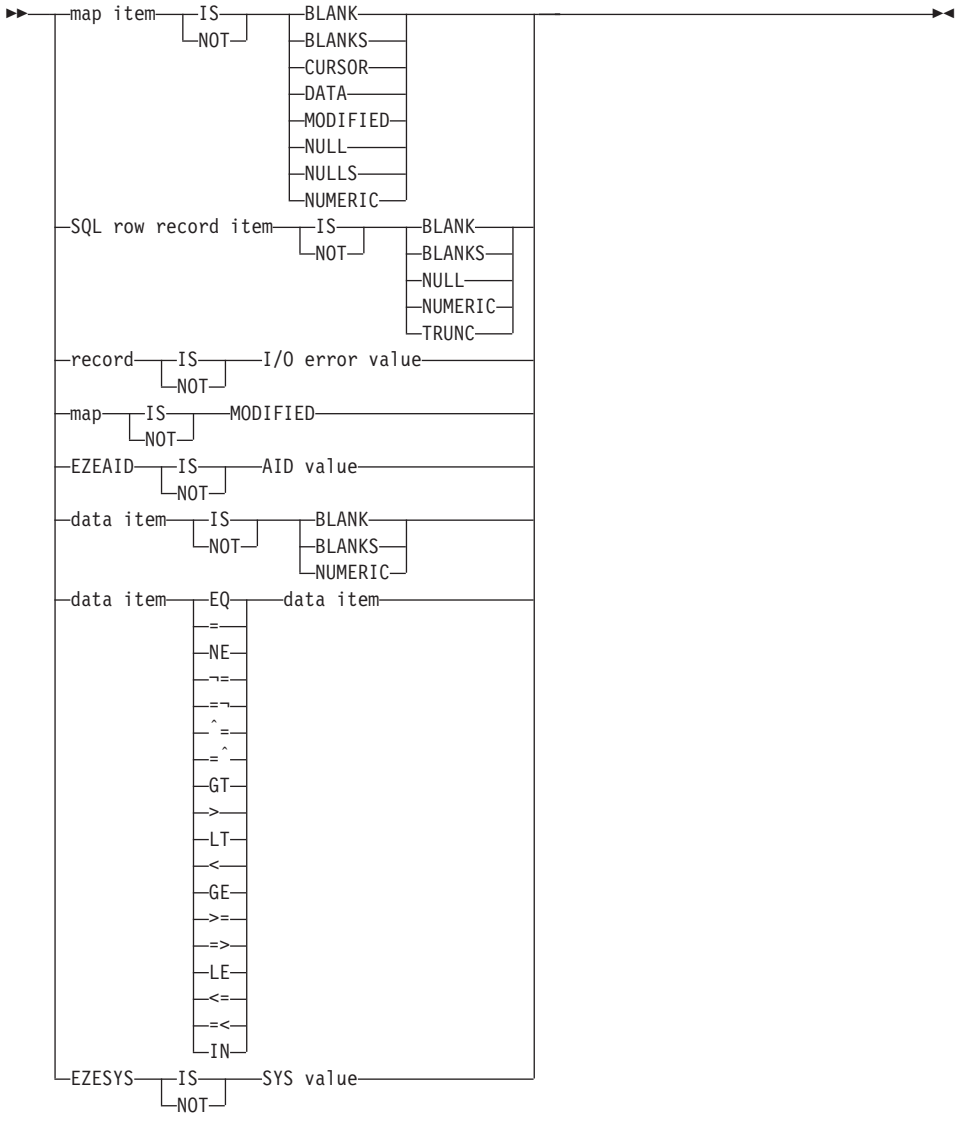
WHILE repeats a block of statements as long as a specific condition or set of conditions is true.



logical expression



condition



Attribute	Description
statement	Any statement or the line that represents the execution of the I/O option within statement definition.
AND, OR	Connectors that can be used to test multiple conditions. With AND, both conditions must be met. With OR, either condition can be met. A combination of AND and OR can be used, but AND is evaluated before OR unless you use parentheses to control the order. Multiple ANDs and ORs can be specified on a single line.

Attribute	Description
map item	Name of a variable field on a map or a map item parameter for a function. A map item can be subscripted, qualified, or both. This comparison is only valid for terminal maps.
IS	Boolean operator that tests true if the specified state is true.
NOT	Boolean operator that tests true if the specified state is false.
BLANK, BLANKS	When used with map items, tests true if either of the following cases are true: • The data received from the display for the specified data item contained all blanks or nulls or both. • The map containing the item has not been conversed since the program started, or since the last SET map CLEAR. When use with non-map items with data type CHA, MIX, or DBCS, tests true if the data item contains all blanks.
CURSOR	Tests that the user left the cursor in the specified data item.
DATA	Tests that there is data other than blanks or nulls within the map item specified. Either the user entered the data or the data was moved to the field before writing to the screen.
MODIFIED	Tests true if data in the variable field has changed. Data is considered changed if any of the following conditions are true: • When specified for a map variable field, data was entered by the program user the last time the map was displayed. • A SET MODIFIED was done prior to the CONVERSE of the map. • The field on the map was defined with a modified data tag (MDT) at map definition time, and this is the first display of the map in the program or the first display of the map after a SET CLEAR. • When specified for a map, tests true if any variable field on the map was changed. **Note:** This saves you from having to test each map field separately.
NULL, NULLS	When specified for map variable field, tests true if either of the following cases are true: • The data entered into the panel for the specified data item contained all nulls or blanks. Nulls are received when the program user presses the Erase EOF key. Note that a true TEST for NULLS does not mean that the field contains nulls internally (it contains blanks). • The map containing the item has not been conversed since the program started, or since the last SET map CLEAR. When use with non-map items with data type CHA, MIX, DBCS, or UNICODE, tests true if the data item contains all blanks.

Attribute	Description
NULL (SQL row record item)	Tests true if the SQL row record item has had no value assigned to the item.
NUMERIC	If the map item or data item type is character or mixed, tests true if the field contains the characters 0 through 9. NUMERIC cannot be used with EZE words.
SQL row record item	Name of a data item in an SQL row record or an SQL item parameter for a function. The name can include a qualifier.
TRUNC	Tests whether a character or a DBCS item in an SQL row record was truncated (nonblank characters deleted on the right) the last time the item value was read from the relational database. Truncation can only occur when the column in the database is longer than the data item. The TRUNC indicator is reset whenever a value is moved to the item, or when the item is set to NULL.
record	Name of a record.
I/O error value	Tests true if the I/O error value specified was returned from the system on the last I/O option that accessed the record. See "I/O error value" on page 368 for more information.
map	Name of a map.
EZEAID	The special function used to test the key that caused the input interrupt from the display.
AID value	Used in testing the state of the EZEAID special function word. See "AID value" on page 364 for more information.
data item	A data item syntactical element. See "Data item" on page 366 for more information.
EQ or =	Boolean operators that test true if data item values are equal.
NE, ¬=, =¬, ^=, or =^	Boolean operators that test true if data item values are not equal. **Note:** The ¬ and ^ symbols are not in the national language syntactic character set, and might not have an equivalent code point across different code pages. If you are exporting your program or generating for machines with differing code pages (in particular, between System/370 host systems and workstations), use NE, not the symbols.
GT or >	Boolean operators that test true if the value of the first data item is greater than the second.
LT or <	Boolean operators that test true if the value of the first data item is less than the second.
GE, >=, or =>	Boolean operators that test true if the value of the first data item is greater than or equal to the second.
LE, <=, or =<	Boolean operators that test true if the value of the first data item is less than or equal to the second.

Attribute	Description
IN	Boolean operator that tests true if the value in the first data item can be found in the array represented by the second data item. If a match is not found, processing skips to the corresponding END statement. **Note:** The value of special function word EZETST is set to 0 if a match is not found. If a match is found, EZETST is set to the index number of the first element of the array that matches the value of the data item. Successive items in the array are compared until a match is found or the end of the array is reached. If the array includes an index, the testing starts there rather than from the first item in the array. If no starting index is given, the test starts with the first item in the array. If the value of the starting index is greater than the number of entries in the array or if no match is found, the test will test false. Comparing against a single data item instead of an array is equivalent to comparing for equal, but is slower and causes setting of EZETST to 0 or 1. It will not be treated as an error. The IN function is similar to the FIND statement in that they both scan for values, but you would use IF or WHILE rather than FIND in the following situations: • IN works with any array, not just a table column. • The search does not have to start at the first entry of the array. • Duplicate values can be found in the array.
EZESYS	The EZESYS test is a runtime test. Generation for a target system will fail if the program includes functions not supported on that system, even if the function is within an IF EZESYS clause that would prevent that function from executing on the target system. To enable generation for the target system to proceed, replace the offending function with a call to a program that performs the function.
SYS value	Used to test the state of the EZESYS special function word. See "SYS value" on page 377 for more information.

Uses

Parentheses can be used to control how conditions are evaluated.

When a conditional expression is nested within parentheses, evaluation proceeds from the least inclusive to the most inclusive part of the expression. The nested expression is evaluated before the expression which contains it. Unless the evaluation order is modified by parentheses, the AND operator is evaluated before the OR operator.

Parentheses can be used to:

- Modify the normal Boolean precedence of operations
- Eliminate ambiguities where operations appear at the same level.

This block of statements controlled by a conditional statement can contain conditional statements. This can continue to a maximum of 15 levels deep.

When WHILE is used in the flow stage of a function and a function name is specified in the block of statements controlled by the WHILE, control is passed to the function named and does not return.

When using the WHILE statement within a function, you may test the map attributes of a parameter item as long as the parameter item has been defined as a map item parameter or the SQL attributes as long as the parameter item has been defined as an SQL item parameter. This capability allows reusable routines to be written to handle the map and SQL item processing.

Target environments for WHILE

ASCII character sets are used in workstation environments. EBCDIC character sets are used in host environments. Differences in collating sequence can cause greater-than or less-than comparisons to have different results in ASCII environments than in EBCDIC environments.

Environment	Compatibility considerations
VM CMS	Uses EBCDIC character sets.
VM batch	Uses EBCDIC character sets.
CICS for MVS/ESA	Uses EBCDIC character sets.
MVS/TSO	Uses EBCDIC character sets.
MVS batch	Uses EBCDIC character sets.
IMS/VS	For a map field to test true when the data entered for a data item contained all blanks, nulls, or a combination of both, the program user must enter at least one blank in the field before pressing the Erase EOF key. If the program user presses Erase EOF without entering one blank in the field, IMS message format services leave Uses EBCDIC character sets.
IMS BMP	Uses EBCDIC character sets.
CICS for VSE/ESA	Uses EBCDIC character sets.
VSE batch	Uses EBCDIC character sets.
CICS for OS/2	Range check comparisons for character data are performed using the ASCII collating sequence.
OS/400	Uses EBCDIC character sets.

Environment	Compatibility considerations
OS/2 (GUI)	Range check comparisons for character data are performed using the ASCII collating sequence. The following are not supported: • WHILE record IS I/O error value • WHILE record NOT I/O value
Windows (GUI)	Same as OS/2 (GUI).
OS/2 (C++)	Uses ASCII character sets.
AIX	Uses ASCII character sets.
HP-UX	Uses ASCII character sets.
CICS for AIX	Uses ASCII character sets.
Windows NT (C++)	Uses ASCII character sets.
Windows NT (Java)	Uses ASCII character sets.
CICS for Windows NT	Uses ASCII character sets.
Solaris	Uses ASCII character sets.
CICS for Solaris	Uses ASCII character sets.
Test Facility	No distinction is made between testing for BLANKS and NULLS. Uses ASCII character sets.

Examples for WHILE

The following examples show you how to use the WHILE statement:

WHILE statement

The following is an example of a WHILE statement:

```
WHILE NUMRECS > 0 AND        /* While there is more data and
  (REQTYPE = 1 OR            /* request type is 1 or
  REQTYPE = 2);              /* request type is 2.
```

WHILE statement summing the elements in an array

The following statements sum the elements in an array, which has 50 entries, but stop early if an entry greater than 9999 is found:

```
MOVE 0 TO TOTAL;
MOVE 1 TO J;
WHILE J LE 50
  AND ARRAY[J] LE 9999;
    TOTAL = TOTAL + ARRAY[J];
    J = J + 1;
END;
```

The two conditions being tested are that J is less than or equal to 50 (that is, the subscript is within the range of the array) and that the entry is less than or equal to 9999. As long as both conditions are met (AND), the entry is added to the sum of the previous entries (TOTAL) and the subscript (J) is incremented. When either condition is not met, the summation stops.

WHILE statement using the IN operation

The IN operation provides a function similar to FIND, but much more powerful. One data item (called ITEM in the example below) is compared to another data item (called LIST in the example below) to ensure they match. The LIST data item must be an array. Successive items in the array are compared until a match is found or the end of the array is reached. If LIST includes an index, the search begins there rather than at the first item in the array. See the following example:

```
MOVE 1 TO START;                          /* Beginning of array
NUMBER-OF-OCCURS = 0;                     /* None found yet
WHILE ITEM IN LIST[START];                /* Find next occurrence
  NUMBER-OF-OCCURS = NUMBER-OF-OCCURS + 1; /* Count number of occurrences
  START = EZETST + 1;                     /* Skip past the one found
END:
```

At this point NUMBER-OF-OCCURS contains the number of times the value in ITEM appears in the array LIST.

The above example is intended to determine the number times a value occurs in an array. The loop will terminate if any of the following conditions occur:

- If ITEM does not appear in LIST
- When START becomes greater than the number of entries in the array LIST.
- When there are no occurrences of the value of ITEM in LIST at or above the entry previously found.

If no starting index is specified, the test starts with the first item in the array. If the value of the starting index is greater than the number entries in the array, the test will test false. It will not be treated as an error.

The following are some advantages of using IN instead of the FIND statement:

- IN works with any array, not just a table column.
- The search does not have to begin at the first array entry.
- Duplicate values can be found in the array.

The following table shows which data items can be compared with each other:

Table 19. Valid data item comparisons

	BIN	CHA	DBCS	HEX	MIX	NUM	NUMC	PACK	PACF	UNICODE
BIN	x					x	x	x	x	
CHA		1		2	1	3				
DBCS			1							
HEX		2		4						
MIX		1			1					
NUM	x	3				x	x	x	x	
NUMC	x					x	x	x	x	
PACF	x					x	x	x	x	
PACK	x					x	x	x	x	
UNICODE										1

Legend:

x Valid data item comparison

1 For CHA to CHA, DBCS to DBCS, MIX to MIX, UNICODE to UNICODE, CHA to MIX, or MIX to CHA comparisons,the shorter item is logically padded on the right with blanks to the length of the longer item. All comparisons are logical comparisons.

2 Valid only if CHA field contains hexadecimal characters (a-f, A-F, 0-9). If a HEX item is compared to a CHA item, the CHA item is converted to HEX format, the shorter field is padded on the right with binary zeros, and a logical comparison is made.

3 Indicates that the data content of the source is validated prior to comparison. If nonnumeric, the program is abnormally terminated. Valid only if the numeric field is defined without decimal positions. The shorter field is padded on the left with zeros.

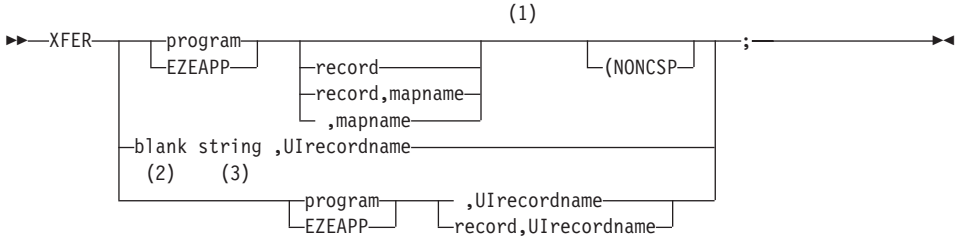
4 If a HEX item is compared to a HEX item, the shorter field is padded on the right with binary zeros to the length of the longer field, and a logical comparison is made.

XFER statement

XFER transfers control to another program or CICS or IMS transaction. When the program to which you are transferring ends, all open files are closed. The current program ends when the transfer occurs and is not resumed when the initiated program or program ends.

You can specify a working storage record, map, or both for the program receiving control.

In a Web transaction program, you can specify a working storage record, a UI record, or both for the program receiving control.



Notes:

1 NONCSP indicates that the program is a non-VisualAge Generator program.

2 EZEAPP can contain a blank string only when EZEAPP is used in conjunction with UI recordname.

3 When EZEAPP does not contain blanks or a non-blank program name is specified, then First UI record should be used in the XFERed to program to receive the data of the specified UI record. In this case, the UI record is sent to be displayed as with XFER with a blank program name, but the data of the optional working storage and the UI record are saved at the server.

Attribute	Description
program	Name of the program to be initiated. If the XFER statement is used in a Web transaction program, a blank program name is only valid if a UI record has been specified.
EZEAPP	A variable name on a transfer statement. This special function word enables you to dynamically change the transferred-to program name within a program. If the XFER statement is used in a Web transaction program, EZEAPP can contain a blank string only if a UI record has been specified.
blank string	The only valid format is one blank space separated by single quotes. For example: `'b' where b is a blank space.` A blank string can be used in place of a program name or EZEAPP. If the XFER statement is used in a Web transaction program, a blank string is only valid if a UI record has been specified.

Attribute	Description
record	Name of any record used in the current program, including working storage. The information in the record is used to initialize the working storage record of the transferred-to program. The maximum size of the record that can be passed is 32767 bytes. Compatible working storage must be defined for the program that is the object of the transfer. If a working storage record is specified on an XFER, only the data in the structure is transferred. Any level-77 data items defined are not transferred. If the receiving working storage is not the same size as the one transferred, the smaller size is used for the transfer. If the receiving area is larger, the primary working storage record of the transferred-to program is initialized based on the type of data (blanks for CHA, DBCS, UNICODE, and MIX, and zero for numeric data). The initialization is done before the transferred record, if any, is moved into the primary working storage record. If the definition of the record specified on the XFER statement is not compatible with the definition of the primary working storage record of the transferred-to program, unpredictable results, including abnormal termination or the display locking up, can occur in the transferred-to program. For example, the following conditions might cause incompatibilities between the two records to occur: • The records differ in length • The field boundaries of the two records do not correspond • The type of data differs (for example, the field is defined as character in the record used on the XFER statement, but the transferred-to program expects the field to be DBCS data). If the XFER statement is used in a Web transaction program, a working storage record can optionally be specified for this parameter. The working storage record is passed to the specified program. The program name cannot be blank.
mapname	Name of a map used in the program. The transferring program must be a main transaction program. The map appears before the transfer to the new program is completed. The new program should use First Map to read the map before the program is run. The map name on the XFER must match the First Map specification for the program that is the object of the transfer. Both programs must use the same map.

Attribute	Description
UIrecordname	If EZEAPP contains blanks or a blank literal is used, then the specified UI record will be sent, as in a CONVERSE, but no state will be saved and the transaction will end. The UI record is sent in all cases regarding program name. If the XFER statement is used in a Web transaction program, a UI record is the only type of record you can specify for this parameter.
NONCSP	Indicates that the program is a non-VisualAge Generator program.

Definition considerations for XFER

XFER from called programs is not supported.

When an XFER statement is used in a Web transaction program:

1. The UI record is sent.
2. The transaction ends.
3. Data is saved or not saved depending on whether the program name is specified:
   - Data is saved if a program name is specified on the XFER statement.

     The data of the optional working storage record and the UI record is saved automatically. Use this style of XFER if the program flow is to be determined completely at the server and the amount of state to save between pages needs to be only what is in the working storage record. If you need to save the state of all data accessible to the program, use CONVERSE instead. Noted that when you use this style of XFER or a CONVERSE the program user will have access to only one page at a time since access to other pages is always controlled by the server. This means the program user will not be able to use the BACK/FORWARD buttons of the browser to get to other pages that may have already been displayed.
   - Data is not saved if a program name is not specified on the XFER statement. Use this style of XFER if the program flow is determined at the client. This means that there are Forms and Program Links on the program users page that allow independent invocation of Web Transactions. Since these programs can be accessed at any time it is not efficient to have state saved for each invocation. This would occur if the results of a client invocation were returned using a CONVERSE or an XFER program. Thus XFER with a blank name means send the results to the browser without saving any state. When application systems are built using a combination of Program Links and Forms on the end user pages together with First UI record and XFER with blank names in the server programs, all pages are freely accessible from the browser at any time using FORWARD/BACK buttons, history logs, etc.

Target environments for XFER

Environment	Compatibility considerations
VM CMS	The transfer is done using the OS XCTL macro. The record and map are passed as parameters. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on transferring program control. The /SYNCXFER generation option controls whether a commit is taken when the XFER is executed for nonsegmented programs. XFER with a map to a non-VisualAge Generator program is not supported. If you are using generated programs as saved segments, the following restrictions apply: • If the saved segment is an initial program then you cannot transfer with a DXFR statement that uses an XCTL or XFER to other programs • You cannot transfer with a DXFR statement that uses an XCTL or XFER to a program that is loaded as a saved segment XFER with a UI record or with a blank program name is not supported.
VM batch	XFER with a map is not supported. Otherwise, same as VM CMS.
CICS for MVS/ESA	The program name or EZEAPP specifies the name of the new transaction to be started. The entry name is the 1- to 4-character name of the transaction to be initiated. The name is truncated to four characters. The transaction is started using a CICS START command. Working storage data is passed in the CICS COMMAREA. The maximum record length is 32763. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on transferring program control. XFER with a map to a non-VisualAge Generator program is not supported.

Environment	Compatibility considerations
MVS/TSO	The transfer is done using the OS XCTL macro. The record, map, and EZEDLPSB, if a PSB was used in the transferring program, are passed as parameters. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on transferring program control. The /SYNCXFER generation option controls whether a commit is taken when the XFER is executed for nonsegmented programs. DL/I calls are supported in the transferred-to program only if there was a PSB specified for the transferring program, and the transferring program does at least one of the following: • Uses CSPTDLI • Uses EZEDLPSB or EZEDLPCB on any statement in the program • Has DL/I databases other than ELAWORK or ELAMSG in the PSB definition XFER with a map to a non-VisualAge Generator program is not supported. XFER with a UI record or with a blank program name is not supported.
MVS batch	The transfer is done using the OS XCTL macro. The record, and EZEDLPSB if a PSB was used in the transferring program, are passed as parameters. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on transferring program control. The /SYNCXFER generation option controls whether a commit is taken when the XFER is executed. DL/I calls and GSAM files are supported in the transferred-to program only if there was a PSB specified for the transferring program, and the transferring program does at least one of the following: • Uses CSPTDLI • Uses AUDIT service routine • Associates at least one file or EZEPRINT with GSAM. • Uses EZEDLPSB or EZEDLPCB on any statement in the program • Has DL/I databases other than ELAWORK or ELAMSG in the PSB definition XFER with a map to a non-VisualAge Generator program is not supported. XFER with a UI record or with a blank program name is not supported.

Environment	Compatibility considerations
IMS/VS	The program name or EZEAPP specifies the name of a new transaction to be started. The entry name is the 1- to 8-character name of the transaction to be initiated. The /SPA generation option for both the transferring and the transferred-to program must be the same. If a map is specified on the XFER, the transferring and the transferred-to programs must share the same map group. The map name can be used in conjunction with a non-VisualAge Generator program. The non-VisualAge Generator program must use the MFS for the map that was created by the VisualAge Generator Developer. XFER cannot be used from batch programs. The maximum size limit for a record passed on an XFER is 32753 bytes. If an input message to a main transaction consists of only the transaction name followed by blanks, the program assumes it is being started with no working storage record being passed. The primary working storage record for the program is initialized based on the data types. The method of passing working storage (SPA, IMS message, or work database) depends on the generation options. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on transferring program control.
IMS BMP	The transfer is done using the OS XCTL macro. The record and the PSB are passed as parameters. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on transferring program control. The /SYNCXFER generation option controls whether a commit is taken when the XFER is executed for nonsegmented programs. XFER with a map is not supported. XFER with a UI record or with a blank program name is not supported.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	Not supported.
CICS for OS/2.	XFER with a UI record or with a blank program name is not supported. Otherwise, same as CICS for MVS/ESA.

Environment	Compatibility considerations
OS/400	Control is passed directly to the program to be initiated using the OS/400 XCTL interface. Working storage is passed as a parameter using a standard system argument list. The program issuing the XFER is removed from the program invocation stack and does not resume control when the initiated program ends. XFER with a map to a non-VisualAge Generator program is not supported. XFER with a UI record or with a blank program name is not supported.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
OS/2 (C++)	The DosExecPgm API is used to transfer control to a non-VisualAge Generator program. The record is passed via a transfer block in shared memory. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on how to transfer control from a VisualAge Generator program to a non-VisualAge Generator program. XFER with a map to a non-VisualAge Generator program is not supported.
AIX	The exec() and fork() system calls are used to transfer control to a non-VisualAge Generator program. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on how to transfer control from a VisualAge Generator program to a non-VisualAge Generator program. XFER with a map to a non-VisualAge Generator program is not supported.
HP-UX	The exec() and fork() system calls are used to transfer control to a non-VisualAge Generator program. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on how to transfer control from a VisualAge Generator program to a non-VisualAge Generator program. XFER with a map to a non-VisualAge Generator program is not supported.
CICS for AIX	Same as CICS for MVS/ESA.
Windows NT (C++)	Same as OS/2 (C++).
Windows NT (Java)	XFER may only be used with UI records and local Java Server Programs.
CICS for Windows NT	Same as CICS for MVS/ESA.

Environment	Compatibility considerations
Solaris	The exec() and fork() system calls are used to transfer control to a non-VisualAge Generator program. Refer to the *VisualAge Generator Client/Server Communications Guide* document for more information on how to transfer control from a VisualAge Generator program to a non-VisualAge Generator program. XFER with a map to a non-VisualAge Generator program is not supported.
CICS for Solaris	Same as CICS for MVS/ESA.
Test Facility	None.

Examples for XFER

The following examples show you how to use the XFER statement.

Transferring control using the XFER statement

The following transfers control to program APPL2 and passes it a working storage record named WSREC:

```
XFER APPL2 WSREC;
```

Using EZEAPP to specify a variable name

The following uses EZEAPP to specify a variable name:

```
MOVE 'APPL2' TO EZEAPP;
XFER EZEAPP;
```

In CICS or IMS, the following transfers a working storage record and a map to a program named APPL3, which is associated with TRX3:

```
MOVE 'TRX3' to EZEAPP;
XFER EZEAPP MYWORK, MYMAP;
```

APPL3 must have MYMAP specified as its first map.

Developing a program for TSO and CICS

If you are developing a program for both the TSO and CICS environments, you can code the following:

```
IF EZESYS IS MVSCICS;
  MOVE 'TRX3' TO EZEAPP;     /* CICS transaction code
  XFER EZEAPP MYWORK, MYMAP;
ELSE;
  XFER APPL3 MYWORK, MYMAP;    /* program name for TSO
END;
```

If you specify the actual program name on the XFER statement for TSO, the generated sample execution CLIST will automatically allocate any data sets for the transferred-to program.

Chapter 11. Special function words

Special function words

Table 20. Special function words

Element	COBOL											GUI		C++									Java	
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	Windows*(Java)	OS/2&Windows(ST)*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	Windows NT	Test Facility
EZEAID	x		x	c		c		x		c	x			c	c	c	c	c	c	x	c	x	c
EZEAPP	x	x	c	x	x	c	x	c	c	c	x			x	x	x	c	x	c	x	c	x	x
EZEBYTES	x	x	x	x	x	x	x	x	x	x	x		x	x	x	x	x	x	x	x	x	x	x
EZECLOS	x	x	x	x	x	c	x	x	x	x	x		x	x	x	x	x	x	x	x	x	x	c
EZECNVCM	x	x	c	x	x	c	x	c	x	c	x			c	c	c	c	c	c	c	c	c	c
EZECOMIT	c	c	c	c	c	i	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
EZECONCT	c	c	c	c	c	c	c	c	c	c	c			c	c	c	c	c	c	x	x	c	x
EZECONV	x	x	x	x	x	x	x	x	x	x				x	x	x	x	x	x	x	x		x
EZECONVT			x					x		x			x	x	x	x	x	x	x	x	x	x	x
EZEC10	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEC11	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEDAY	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEDAYL	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEDAYLC	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEDEST	c	c	c	c	c	c	c	c	c	c	c			c	c	c	c	c	c	c	c	x	c
EZEDESTP	c	c	c	c	c	c	c	c	c	c	c			c	c	c	c	c	c	c	c		c
EZEDLCER (DL/I)			x	c	c	c	c	x	c	c													c
EZEDLCON (DL/I)			x	c	c	c	c	x	c	c													c
EZEDLDBD (DL/I)			x	x	x	x	x	x	x														x

Table 20. Special function words (continued)

Element	COBOL											GUI		C++								Java	Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	Windows*(Java)	OS/2&Windows(ST)*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	Windows NT	
EZEDLERR (DL/I)			x	x	x	x	x	x	x														x
EZEDLKEY (DL/I)			x	x	x	x	x	x	x														c
EZEDLKYL (DL/I)			x	x	x	x	x	x	x														x
EZEDLLEV (DL/I)			x	x	x	x	x	x	x														x
EZEDLPCB (DL/I)			c	c	c	c	c	c	c														c
EZEDLPRO (DL/I)			x	x	x	x	x	x	x														x
EZEDLPSB (DL/I)	i	i	c	c	c	c	c	c	c	c	i			i	i	i	c	c	c	i	c	i	c
EZEDLRST (DL/I)			x	c	c	c	c	x	c	c													c
EZEDLSEG (DL/I)			x	x	x	x	x	x	x														x
EZEDLSSG (DL/I)			x	x	x	x	x	x	x														x
EZEDLSTC (DL/I)			x	x	x	x	x	x	x														x
EZEDLTRM (DL/I)			c	x	i	c	i	c	i	c													c
EZEDTE	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEDTEL	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEDTELC	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEFEC	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEFLO	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x	c
EZEG10	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Table 20. Special function words (continued)

Element	COBOL											GUI		C++								Java	
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	Windows*(Java)	OS/2&Windows(ST)*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	Windows NT	Test Facility
EZEG11	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZELOC	c	c	x	c	c	c	c	x	c	x		x	x				x		x		x	x	x
EZELTERM	c	c	c	c	c	c	c	c	c	c	c			c	c	c	c	c	c	c	c	c	x
EZEMNO	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x		x
EZEMSG	x	x	x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x	x		x
EZEOVER	x	x	x	x	x	x	x	x	x	x	x	x	x	c	c	c	c	c	c	c	c	x	c
EZEOVERS	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEPURGE	i	i	x	i	i	i	i	x	i	x							x		x		x		
EZERCODE	x	x	c	x	x	c	x	c	x	c				c	c	c	c	c	c	c	c	i	x
EZEREPLY	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEROLLB	c	c	c	c	c	c	c	c	c	c	x	c	c	c	c	c	c	c	c	c	c	c	
EZERTN	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
EZERT8	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	
EZESEGM	c	i	x	c	i	c	i	x	i	x				c	c	c	c	c	c	c	c	c	x
EZESEGTR	i	i	c	i	i	c	c	c	i	c				c	c	c	c	c	c	c	c	i	i
EZESQCOD (SQL)	c	c	c	c	c	c	c	c	c	c	c			c	c	c	c	c	c	c	c	c	c
EZESQISL (SQL)	c	c	i	i	i	i	i	c	c		i			i	i	i	i	i	i	i	i	i	
EZESQLCA (SQL)	c	c	c	c	c	c	c	c	c	c	c			c	c	c	c	c	c	c	c	c	x
EZESQRD3 (SQL)	c	c	c	c	c	c	c	c	c	c	c			c	c	c	c	c	c	c	c	c	x
EZESQRRM (SQL)	c	c	c	c	c	c	c	c	c	c	c			c	c	c	c	c	c	c	c	c	x
EZESQWN1 (SQL)	c	c	c	c	c	c	c	c	c	c	c			c	c	c	c	c	c	c	c	x	x

Table 20. Special function words (continued)

Element	COBOL											GUI		C++								Java		Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	Windows*(Java)	OS/2&Windows(ST)*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	Windows NT	
EZESQWN6 (SQL)	c	c	c	c	c	c	c	c	c	c	c			c	c	c	c	c	c	c	c	c	x
EZESYS	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZETIM	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZETST	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEUSR	c	c	c	c	c	c	c	c	c	c	x			c	c	c	c	c	c	c	c	c	c
EZEUSRID	c	c	c	c	c	c	c	c	c	c	x			c	c	c	c	c	c	c	c	c	c
EZEWAIT	x	x	x	x	x	i	x	x	x	x	x			x	x	x	x	x	x	x	x	x	

Note: *Includes Windows 95 and Windows 98, Windows NT, and Windows 2000.

Legend: In this table, the following characters are used to indicate the level of support:

x	Supported
c	Supported with compatibility considerations
i	Ignored.
blank	Not supported
ST	Smalltalk

EZEAID

EZEAID indicates the function key the program user pressed during a map I/O operation.

EZEAID is reset on every CONVERSE I/O option. If a first map is not specified, EZEAID is set to Enter until the first CONVERSE I/O option for a map occurs.

Uses

You can use EZEAID to test for the following values:

- PA (any program access, or PA, key), PA1, PA2, and PA3
- PF (any function key), PF1 through PF24
- Enter
- BYPASS (any bypass edit function key)

Use EZEAID as an operand of the following statement types:
- IF statement
- WHILE statement
- TEST statement

The characteristics of EZEAID follow:

Data type
Character

Data length in bytes
1

Value saved across segments
No

Target environments for EZEAID

Environment	Compatibility considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	PF6 is reserved for a panel recovery function. If the program user presses PF6, it is treated as pressing the Clear key. The AID value is not passed back to the program.
MVS batch	Not supported.
IMS/VS	PA1, PA2, and PA3 are reserved for paging by IMS/VS. If your installation uses PF12 for the IMS local copy function, PF12 cannot be used. If these keys are pressed, no AID value is passed to the program.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.

Environment	Compatibility considerations
CICS for OS/2	Mapping the personal computer keyboard keys to 3270 keys is defined in the CICS OS/2 Workstation Setup (WSU) table. The following is the default mapping: **Program Function Key Personal Computer Key** PF1 - PF12 F1 - F12 PF13 - PF24 Alt+F1 - Alt+F12 PA1 - PA3 Ctrl+F1 - Ctrl+F3 Refer to the *CICS for OS/2 System and Application Guide* for information on how to modify the WSU table to change the key mapping. Closing the Map Monitor during CONVERSE is the same as pressing PA2 which is the default bypass edit key. Thus, after closing Map Monitor during a CONVERSE, EZEAID will have a value of PA2.
OS/400	None.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	The following is the default mapping: **AID Personal Computer Key** PF1 - PF12 F1 - F12 PF13 - PF14 Shift+F1 - Shift+F12 PA1 - PA3 Ctrl+F1 - Ctrl+F3
AIX	Same as OS/2 (C++).
CICS for AIX	Mapping the AIX terminal keyboard keys to 3270 keys is defined in the 3270keys file, which is part of TCP/IP. The default values depend on the type of terminal or session you are using to run CICS for AIX transactions. For more information, refer to "3270keys File Format for TCP/IP using InfoExplorer", in the AIX online help facility.
HP-UX	Same as OS/2 (C++).
Solaris	None.

Environment	Compatibility considerations
CICS for Solaris	Mapping the AIX terminal keyboard keys to 3270 keys is defined in the 3270keys file, which is part of TCP/IP. The default values depend on the type of terminal or session you are using to run CICS for AIX transactions. For more information, refer to "3270keys File Format for TCP/IP using InfoExplorer", in the AIX online help facility.
Windows NT (C++)	Same as OS/2 (C++).
Windows NT (Java)	None.
CICS for Windows NT	Mapping the Windows NT terminal keyboard keys to 3270 keys is defined in the 3270keys file, which is part of TCP/IP. The default values depend on the type of terminal or session you are using to run CICS for Windows NT transactions. For more information, refer to "3270keys File Format for TCP/IP using InfoExplorer", in the Windows NT online help facility.
Test Facility	The test facility supports the default CICS OS/2 mapping.

Example for EZEAID

```
IF EZEAID IS PA1;
END;
```

EZEAPP

EZEAPP can be used to change the name of the program or non-VisualAge Generator program to which you want to transfer.

EZEAPP is used with the DXFR or XFER statements to change the transferred-to program name while the program is running. EZEAPP is set to the program name before the transfer statement runs. EZEAPP cannot contain DBCS literals. The contents of EZEAPP are not checked when the transfer statement runs.

In a Web transaction program, EZEAPP can contain a blank when a UI record is specified on the XFER statement.

Uses

You can use EZEAPP as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand in a MOVE or assignment statement
- The program operand in a DXFR or XFER statement
- Data item 1 or 2 in a RETR statement

- The data item of an IF or WHILE statement
- The data item of a FIND statement
- Operand of a TEST statement

The characteristics of EZEAPP follow:

Data type
Character

Data length in bytes
8

Value saved across segments
Yes

Definition considerations for EZEAPP

When EZEAPP is a target operand, the contents of EZEAPP are automatically folded to uppercase. Therefore, any subsequent use of EZEAPP will involve the folded version.

Target environments for EZEAPP

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	When EZEAPP is used with an XFER statement, EZEAPP contains the name of the new transaction to be started, not the program name. Only the first 4 bytes of the name are used. EZEAPP is not folded in CICS environments.
MVS/TSO	None.
MVS batch	None.
IMS/VS	When EZEAPP is used with an XFER statement, EZEAPP contains the name of the new transaction to be started, not the program name.
IMS BMP	None.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	Transfers using XFER are not supported.
CICS for OS/2	Same as CICS for MVS/ESA.
OS/400	None.
Windows & OS/2 Smalltalk (GUI)	Not supported.

Environment	Compatibility considerations
Windows Java (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
Solaris	None.
CICS for Solaris	Compatibility considerations.
CICS for AIX	Same as CICS for MVS/ESA.
Windows NT (C++)	None.
Windows NT (Java)	None.
CICS for Windows NT	Same as CICS for MVS/ESA.
Test Facility	None.

Example for EZEAPP

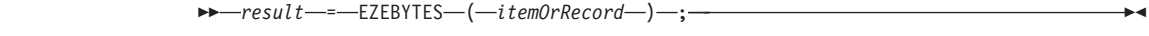
The following example puts a program name into EZEAPP:

```
MOVE 'APPL' TO EZEAPP;
DXFR EZEAPP;
```

EZEBYTES

EZEBYTES returns the length of an item or record in bytes.

Uses



Attribute	Description
result	Full-word integer (bin, bytes=4) which on return contains the length of *itemOrRecord*.
itemOrRecord	Any data item or record part type.

Target environments for EZEBYTES

Supported in all environments without compatibility considerations.

Example for EZEBYTES

```
EZEREPLY = 0;
RESULT = EZEBYTES (Recd1);
```

EZECLOS

EZECLOS immediately ends the program. If the program is a called program, EZECLOS returns control to the calling program. EZECLOS is the default flow for the last main function in a program.

Uses

You can use EZECLOS as any of the following:

- The name of a function error routine
- The true or false operand of a TEST or FIND statement in a function or flow
- A function invocation statement

Target environments for EZECLOS

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	For a segmented program that runs with a scratchpad area, EZECLOS causes the IMS/VS conversation to end.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
Windows & OS/2 Smalltalk (GUI)	None.
Windows Java (GUI)	None.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.

Environment	Compatibility considerations
Solaris	None.
CICS for Solaris	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	None.
CICS for Windows NT	None.

Test Facility	If EZEFLO or EZECLOS is encountered and Exit breakpoints are set on any of the functions that are currently listed in the Execution Stack Monitor, the following status message is displayed in the status area of the Test Monitor:
	`Exit Breakpoints exist`
	The break in execution will occur on the EZEFLO or EZECLOS statement.
	If EZECLOS is encountered and Exit breakpoints are not set on any of the functions that are currently listed in the Execution Stack Monitor, but the current program uses Exit breakpoints, the following status message is displayed in the status area of the Test Monitor:
	`Break on program - program_name`
	The break in execution will occur on the EZECLOS statement.

Example for EZECLOS

```
TEST EZEAID PF1 EZECLOS, FALSEGP;
```

EZECNVCM

EZECNVCM is a switch used to control whether data is automatically committed for every CONVERSE I/O option. When EZECNVCM is set to 1, EZECOMIT is called during every CONVERSE. The default setting of EZECNVCM is 0 if non-segmented and 1 if segmented. The program can change EZECNVCM at any time.

Uses

You can use EZECNVCM as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand in a MOVE or assignment statement
- An occurrence operand in a MOVEA statement
- Data item 1 in a RETR statement

- A data item in an IF or WHILE statement
- A data item in a FIND statement

The characteristics of EZECNVCM follow:

Data type
Numeric

Data length in bytes
1

Value saved across segments
Yes

Definition considerations for EZECNVCM

When EZECNVCM is set to 1, EZECOMIT is automatically called during every CONVERSE function following terminal write, but before terminal read. This commits data changes to files or databases and logs terminal output at the same time. When EZECNVCM is set to 0, a commit is done on the CONVERSE only if the program is running in segmented mode at the time of the CONVERSE.

Target environments for EZECNVCM

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	EZECNVCM is ignored on a segmented CONVERSE. The CONVERSE marks the end of a segment; a commit is always done at the end of a segment.
MVS/TSO	None.
MVS batch	None.
IMS/VS	The value of EZECNVCM is ignored. In IMS, each CONVERSE is the end of a segment; a commit is always done at the end of a segment.
IMS BMP	None.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	None.
CICS for OS/2	Same as CICS for MVS/ESA.
OS/400	None.
Windows & OS/2 Smalltalk (GUI)	Not supported.

Environment	Compatibility considerations
Windows Java (GUI)	Not supported.
OS/2 (C++)	Commits changes to relational databases. Files are not affected.
AIX	Same as OS/2 (C++).
HP-UX	Same as OS/2 (C++).
Solaris	Compatibility considerations.
CICS for Solaris	Compatibility considerations.
CICS for AIX	Commits changes to relational databases and files defined as recoverable resources to CICS for AIX.
Windows NT (C++)	Same as OS/2 (C++).
Windows NT (Java)	Same as OS/2 (C++).
CICS for Windows NT	Commits changes to relational databases and files defined as recoverable resources to CICS for Windows NT.
Test Facility	If the program is running under the test facility, data is committed only when a map appears.

Example for EZECNVCM

```
MOVE 1 TO EZECNVCM;
```

EZECOMIT

EZECOMIT calls services to save recoverable file, database, and message queue updates since the last commit.

If the program is running in a transactional environment (CICS, IMS, or OS/400), VAGen issues an environment commit that performs a two-phase commit coordinated across all resource managers. In non-transaction environments, VAGen performs a single phase commit calling each recoverable resource manager separately.

The scan position is lost and update locks are released for any files or databases affected by the EZECOMIT. An exception to this occurs when using Declare Cursor With Hold. The WITH HOLD option is not supported for ODBC. When using ODBC, all open cursors are closed on an EZECOMIT and all statements must be re-prepared.

Uses

You can use EZECOMIT as the function name in a function invocation statement.

Always use EZECOMIT in GUIs prior to completing processing of any event that results in relational database updates being made by local DLLs called by the GUI or calls to a remote server where client controlled unit of work is specified for the server call.

Definition considerations for EZECOMIT

You should consider the need for using EZECOMIT and the implications of using EZECOMIT with message queues. The following sections examine when the use of EZECOMIT is unnecessary due to implicit commit situations and definition considerations for EZECOMIT when message queues are involved.

EZECOMIT and implicit commit situations

Note: You might not need to use EZECOMIT if you consider the following implicit commit situations. If you can take advantage of these implicit commit situations, you could enhance performance by not using EZECOMIT explicitly.

A commit point is taken on any of the following:

- When a program calls either the EZECOMIT or COMMIT service routine.

  For VM CMS, VM batch, MVS/TSO, MVS batch, and VSE batch, VisualAge Generator programs that do not use DL/I issue a commit point only if the program has made changes to an SQL table. A commit point does not occur for changes to an SQL table made by a non-VisualAge Generator program.

  For IMS/VS and transaction-oriented IMS BMP programs (programs that scan a serial file associated with the I/O PCB), EZECOMIT is ignored. A commit point occurs whenever there is a get unique to the I/O PCB.
- When the top-level program in a run unit ends successfully.

  For VM CMS, VM batch, MVS/TSO, MVS batch, IMS BMP, and VSE batch, a run unit consists of all VisualAge Generator programs and non-VisualAge Generator programs that transfer control among themselves using an XFER, DXFR, or CALL statement. For non-VisualAge Generator programs, this also includes any transfer that uses an OS XCTL macro or CALL statement.

  For CICS or IMS/VS, a run unit is equivalent to a single transaction and consists of all VisualAge Generator programs and non-VisualAge Generator programs that transfer control among themselves using a DXFR or CALL statement. For non-VisualAge Generator programs, this also includes any transfer that uses a CALL statement, a CICS command.
- When a program uses a CONVERSE I/O option and any of the following is set to 1:
  - EZESEGM special function word (segmented mode).

    The EZESEGM special function word defaults to 1 if the program is defined as segmented (in environments that support segmented mode).
  - EZECNVCM special function word (CONVERSE commit)

  - EZEDLTRM special function word (end PSB at CONVERSE) if the program uses DL/I.

  The best time for a commit point to occur is after terminal output and before the next terminal input. A commit point at terminal I/O synchronizes updates to the database and confirmation messages to the program user.
- On a XFER statement, unless the /NOSYNCXFER option was specified for VM CMS, VM batch, MVS/TSO, MVS batch, or batch-oriented IMS BMP programs.
- For IMS/VS and transaction-oriented IMS BMP programs, when a program does a successful GET UNIQUE to the I/O PCB
- For CICS, when a transfer using a DXFR statement occurs, a PSB is scheduled, and one of the following occurs:
  - Transfer to a non-VisualAge Generator program
  - The /SYNCDXFR generation option is specified for the transferred-from program
  - The /NOSYNCDXFR generation option is specified for the transferred-from program and different PSB names were identified in the program specifications for the two programs.
- For CICS, when a called DL/I program returns to the calling program, the PSB was not passed using the EZEDLPSB special function word, and PCBs were not passed using the EZEDLPCB special function word.

EZECOMIT and message queues

When you use EZECOMIT with message queue records, note the following:

- Message queue updates are recoverable only if the **Include message in transaction** option is selected in message queue record definition.
- Both message SCANs and ADDs are affected by commit and rollback for recoverable messages. If a rollback is issued following a SCAN for a recoverable message, the message is placed back on the input queue so that the input message is not lost when the transaction fails to complete successfully.

Target environments for EZECOMIT

Environment	Compatibility considerations
VM CMS	EZECOMIT commits changes to relational databases. EZECOMIT results in an SQL COMMIT WORK if the program has issued SQL requests.
VM batch	Same as VM CMS.

Environment	Compatibility considerations
CICS for MVS/ESA	EZECOMIT results in a CICS SYNCPOINT, which commits changes to relational databases, DL/I databases, and files defined as recoverable resources to CICS. The end of a segment (issuing a CONVERSE when running in segmented mode) has the same effect as EZECOMIT. Remote called batch program (programs that reside on a different system than the invoking program) can invoke EZECOMIT. If ECI_NO_EXTEND was specified, commit will work. If ECI_EXTEND was specified, a runtime error message is issued that commit failed with INVREQ.
MVS/TSO	EZECOMIT commits changes to relational and DL/I databases. EZECOMIT results in an SQL COMMIT WORK if the program has issued SQL requests. A DL/I CHKP (checkpoint) call is issued if the program has issued DL/I requests. The contents of special function word EZEDLPSB are used as the checkpoint identifier on the CHKP call.
MVS batch	If the batch has not specified a PSB but has issued SQL requests, an SQL COMMIT WORK is issued. If the program has a PSB specified, invoking EZECOMIT results in a DL/I basic CHKP call, which commits changes to databases. The contents of the special function word EZEDLPSB are used as the checkpoint identifier on the CHKP call. GSAM files are not recoverable when used with basic CHKP. To make GSAM files recoverable, use CSPTDLI for symbolic checkpoint instead of EZECOMIT.
IMS/VS	EZECOMIT is ignored in this environment. Commit processing is done by the system at the following points: • On each CONVERSE • When a transaction program finishes: – At the end of the last main function – Executing an EZECLOS – Executing an XFER • When a batch program scans a serial file assigned to the I/O PCB for the first segment of the next message • When a batch program uses CSPTDLI to issue a CHKP function or a get unique call to the I/O PCB Changes to all databases and serial files are committed.

Environment	Compatibility considerations
IMS BMP	For a batch-oriented BMP program that does not scan a serial file associated with the I/O PCB, invoking EZECOMIT results in a DL/I basic CHKP call, which commits changes to all databases. The contents of the special function word EZEDLPSB are used as the checkpoint identifier on the CHKP call. GSAM files are not recoverable when used with basic CHKP. To make GSAM files recoverable, use CSPTDLI for symbolic checkpoint instead of EZECOMIT. EZECOMIT is ignored for transaction-oriented BMP programs in which the program scans a serial file associated with the I/O PCB. For these programs the commit is done by the system each time a SCAN is issued for a serial file assigned to the I/O PCB to obtain the first segment of a message or when CSPTDLI is used to request a CHKP function.
CICS for VSE/ESA	Remote server programs (programs that reside on a different system than the invoking program) can issue EZECOMIT only if the linkage table specified server unit of work for the program. Otherwise, same as CICS for MVS/ESA.
VSE batch	If the program has not specified a PSB but has issued SQL requests, an SQL COMMIT WORK is issued. If the program has a PSB specified, invoking EZECOMIT results in a DL/I basic CHKP call, which commits changes to databases. The contents of the special function word EZEDLPSB are used as the checkpoint identifier on the CHKP call.
CICS for OS/2	EZECOMIT results in a CICS SYNCPOINT. The CICS SYNCPOINT commits changes made to DB2 databases and files defined as recoverable files to CICS. The SYNCPOINT also commits changes to databases and recoverable files on a CICS host when the changes are made by a remote called batch program that is called by the program invoking EZECOMIT. CICS coordinates the host and workstation commit functions. The SQL COMMIT WORK commits changes made to relational databases. Remote called batch programs (programs that reside on a different system than the invoking program) can invoke EZECOMIT. If ECI_NO_EXTEND was specified, commit will work. If ECI_EXTEND was specified, a runtime error message is issued that commit failed with INVREQ. Files generated with file type OS2COBOL are not recoverable resources and are not affected by EZECOMIT.

Environment	Compatibility considerations
OS/400	An implicit commit is issued when a MAIN type program ends, which constitutes the end of a run unit. The most common instance of this is when the MAIN program returns to the non-VisualAge Generator program from which it was started. Another instance of an implicit commit is on an XFER. A DXFR does not cause a commit. Invoking EZECOMIT results in an SQL COMMIT WORK if the program has issued SQL requests. Invoking EZECOMIT results in the equivalent of an OS/400 COMMIT command, if the program has not issued SQL requests.
Windows & OS/2 Smalltalk (GUI)	EZECOMIT results in a commit of relational database changes made by generated C++ programs called locally plus any recoverable resource changes made by remote server programs called using client controlled unit of work in the linkage table. A two-phase commit is used only if all the servers were called using a transaction manager under the same transaction.
WindowsJava (GUI)	Same as Windows & OS/2 Smalltalk (GUI)..
OS/2 (C++)	Commits changes to relational databases and changes made to remote server programs called using client controlled unit of work. Single phase commit is used where there are multiple resource managers.
AIX	Same as OS/2 (C++).
CICS for AIX	Commits changes to relational databases, files defined as recoverable to CICS, and changes made by remote server programs defined in the linkage table as using client controlled unit of work. The commit can be set up as a two-phase commit controlled by CICS.
HP-UX	Same as OS/2 (C++).
Solaris	Same as OS/2 (C++).
CICS for Solaris	Commits changes to relational databases, files defined as recoverable to CICS, and changes made by remote server programs defined in the linkage table as using client controlled unit of work. The commit can be set up as a two-phase commit controlled by CICS.
Windows NT (C++)	Commits changes to relational databases and files defined as recoverable resources to Windows NT.
Windows NT (Java)	Same as OS/2 (C++).
CICS for Windows NT	Same as CICS for AIX.
Test Facility	EZECOMIT is ignored in the IMS/VS DL/I execution environment. The call to EZECOMIT is ignored, but the test continues.

Example for EZECOMIT

```
EZEREPLY = 0;
EZECOMIT();
```

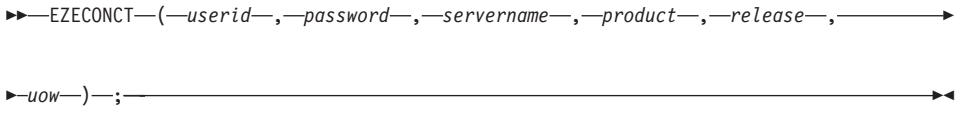
EZECONCT

EZECONCT controls the database unit of work in VisualAge Generator programs. EZECONCT enables a program to connect, disconnect, or activate database connections.

Uses

You can use EZECONCT as the function name in a function invocation statement.

The following is the calling sequence for EZECONCT:



userid A database user identifier (8-byte CHA data item). Refer to "Authorization considerations", in the *VisualAge Generator Design Guide* document, for more information on options for specifying a database userid and password for workstation programs.

password
A database password (8-byte CHA data item).

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

For MVS, this argument contains one the following:
- Blanks
- RESET
- The name of the program server to receive the SQL requests.

For other systems, it contains the name of the program server for the database requests.

Table 21. Arguments allowed on invocation of EZECONCT

Arguments	Connection
`servername` contains RESET	Reconnect to default database (DB2 CONNECT RESET); fill `product` and `release` with information about the server

Table 21. Arguments allowed on invocation of EZECONCT (continued)

Arguments	Connection
`servername` nonblank, but does not contain RESET	Connect to specified server (DB2 CONNECT TO); fill `product` and `release` with information about the server
`servername` is blank	Perform a query for product and release level of the server (DB2 CONNECT); fill `product` and `release` with information about the server

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

This argument is optional, but must be specified if `release` is specified.

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

This argument is optional, but must be specified if `uow` is specified.

uow The unit of work connection option (CHA item or literal).

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

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

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

Remote unit of work must be used if the database managers are at the following levels:
- DB2 Version 2
- DB2/6000 Version 1
- DataJoiner Version 1
- Oracle

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

Dxy Type 2 Connect, Distributed Unit of Work.

Type 2 is only supported for DB2 and DataJoiner.

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

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

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

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

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

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

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

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

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

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

DISC Disconnect from the database identified in `servername`.

DCURRENT
Disconnect from the currently connected database.

DALL Disconnect from all currently connected databases.

SET Set connection to dormant database connection.

Definition considerations for EZECONCT

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

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

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

- Issue all EZECONCT, EZEROLLB, and EZECOMIT requests from a program running under the test facility, not from the generated C++ program.
- Set the default database name in the VAGen-SQL tab of the VisualAge Preferences window. Do not set the VisualAge Generator environment variables EZERSQLDB or FCWDBNAME_<applname>.

Following the EZECONCT invocation, if EZEFEC is set to 1, the EZESQ special function words are set to values returned by the CONNECT statement.

Default Database Connections

If EZECONCT is not used, the default database connection is a Type 1 (remote unit of work) connection. The specification of the default server name varies with the environment.

The test facility checks the following for the server name in this order:

- Database name in VisualAge Generator SQL Preferences
- If no database name is passed to DB2/2 on a default database connection, DB2/2 checks the DB2/2 environment variable DB2DFTDB for the database name.

CICS for OS/2 programs check for the server name in this order:

- ELARTRDB_tttt where `tttt` is the CICS transaction identifier.
- EZERSQLDB environment variable
- If no database name is passed to DB2/2 on a default database connection, DB2/2 checks the DB2/2 environment variable DB2DFTDB for the database name.

Generated C++ programs check for the server name in this order:

- FCWDBNAME_applname where `applname` is the name of the program issuing the first SQL request
- EZERSQLDB environment variable

- If no database name is passed to DB2/2 on a default database connection, DB2/2 checks the DB2/2 environment variable DB2DFTDB for the database name.
- If the server name is not found in the above checks, DB2 checks the DB2 environment variable DB2DFTDB (native C++ programs) or the default database defined to CICS for AIX region (recommended for best performance for CICS for AIX programs).

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

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

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

Target environments for EZECONCT

Environment	Compatibility considerations
VM CMS	CONNECT RESET and UOW (the unit of work connection option) are not supported. The password argument is supported.
VM batch	Same as VM CMS.
CICS for MVS/ESA	Connection functions are supported by DB2 Version 2 Release 3 or later. The password argument is ignored. UOW (the unit of work connection option) is not supported.
MVS/TSO	Same as CICS for MVS/ESA.
MVS batch	Same as CICS for MVS/ESA.
IMS/VS	Same as CICS for MVS/ESA.
IMS BMP	Same as CICS for MVS/ESA.
CICS for VSE/ESA	Connection to a different database is not supported. Only one database can be used for each transaction and that database is associated with the transaction in the CICS tables. CONNECT RESET is not supported. UOW (the unit of work connection option) is not supported. The password argument is ignored.

Environment	Compatibility considerations
VSE batch	Connection to a different database requires DB2/VSE 3.4 or later. CONNECT RESET is not supported. The password argument is supported. UOW (the unit of work connection option) is not supported.
CICS for OS/2	If ELARTRDB_tttt or EZERSQLDB is used to specify the database name, the first SQL program in the run-unit will connect to the specified database while the program is initialized. In addition, following an EZECONCT RESET, the program will immediately reconnect to this database. If the DB2/2 SQL default database is being used, DB2/2 will perform an implicit connect when the run-unit performs its first SQL statement. The same applies following an EZECONCT RESET. **Note:** Since DB2/2 is handing the connection, the default access mode specified in EZERSQLUS will not be used. All databases connected using implicit connect are connected in SHARED mode. For more information about implicit connect, see your DB2/2 documentation. DB2/2 Version 2.1 is required for password support. UOW (the unit of work connection option) is not supported.
OS/400	The password argument is ignored. UOW (the unit of work connection option) is not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	DB2/2 Version 2.1 or later is required for password support.
AIX	The password argument is supported.
HP-UX	The password argument is supported.
CICS for AIX	The password argument is ignored.
Solaris	None.
CICS for Solaris	None.
Windows NT (C++)	The password argument is supported.

Environment	Compatibility considerations
Windows NT (Java)	When connecting to a JDBC database via EZECONCT, the physical database name is found by looking up the property vgj.jdbc.database.xxxx, where xxxx is the name of the server specified on the EZECONCT call. If this property is not defined, the server name specified on the EZECONCT call is used as-is.
CICS for Windows NT	The password argument is ignored.
Test Facility	None.

Example for EZECONCT

```
EZECONCT('scalia','justs4u','dojhost','goodz','4.0','D1C');
```

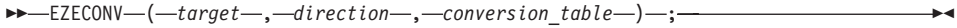
EZECONV

EZECONV converts data between EBCDIC (host) and ASCII (workstation) formats, or performs code page conversion within a single format.

Uses

You can use EZECONV as the function name in a function invocation statement.

The following is the calling sequence for EZECONV:



target The name of the record, map, or data item that has the format you want to convert. The data is converted in place based on the item definition of the lowest-level items (items with no substructure) in the target object.

Variable length records are converted only for the length currently set for the record in the number of occurrences item for variably occurring records, or in the record length or variable length item for other types of records. A conversion error occurs and the program ends if the variable length record ends in the middle of a numeric field or a DBCS character.

direction ('R' or 'L')
An optional character literal identifying the direction of the conversion. If you specify 'R', the data is assumed to be in remote format and is converted to local format. If you specify 'L', the data is assumed to be in local format and is converted to remote format as defined in the conversion table. 'R' is the default. The 'R' and the 'L' must be surrounded by quotation marks.

Direction is required if you specify `conversion_table`.

conversion_table
An optional 8-character data item or literal specifying the name of the conversion table you want to use for data conversion. The default is the conversion table associated with the national language code specified when the program was generated.

Definition considerations for EZECONV

You can use the linkage table to request that automatic data conversion be generated for remote calls, CREATX requests, and file I/O requests.

Automatic conversion is always performed using the data structure defined for the argument being converted. Do not request automatic conversion if an argument has multiple formats. For arguments with multiple formats, code the program to explicitly call EZECONV with redefined record definitions that correctly map the current values of the argument.

For more information, refer to the section on converting data in the *Design Guide* document.

Target environments for EZECONV

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	None.

Environment	Compatibility considerations
AIX	None.
HP-UX	None.
CICS for AIX	None.
Solaris	None.
CICS for Solaris	None.
Windows NT (C++)	None.
Windows NT (Java)	Not supported.
CICS for Windows NT	None.
Test Facility	None.

Example for EZECONV

In the following example, MY_RECORD is defined as a VisualAge Generator record and MY_CONV_TABLE is an 8-byte character data item containing the name of the conversion table.

```
EZECONV(MY_RECORD,'L',MY_CONV_TABLE);
```

EZECONVT

EZECONVT contains the name of the conversion table used to dynamically convert data in an argument or record structure on CALL or CREATX requests to programs on remote systems or on file I/O requests to files at remote locations. The conversion occurs when the data is being moved between EBCDIC-based and ASCII-based systems or between systems that use different code pages.

Conversion is bypassed at run time if EZECONVT is blank.

Uses

You can use EZECONVT as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand of a MOVE or assignment statement
- Data item 1 or 2 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand on the TEST statement

The characteristics of EZECONVT follow:

Data type: Character

Data length in bytes: 8
Value saved across segments: Yes

Definition considerations for EZECONVT

You should use EZECONVT to switch conversion tables in a program or to turn data conversion on or off in a program. The value in EZECONVT is used only when EZECONVT is specified as the data conversion table name in the linkage table.

You can use the linkage table to specify that a CICS program invoked by a CALL or CREATX statement or a CICS VSAM file or transient data queue is located at a remote system. If you specify EZECONVT as the conversion table name for a remote function in the linkage table, and EZECONVT contains a conversion table name, automatic data format conversion is performed when the remote CALL statement, CREATX service routine, or file I/O function is processed. If EZECONVT is blank, no conversion is performed.

Conversion is performed on the system that originates the function based on the description of the arguments defined in the originating program. When you define multiple levels of a record structure, conversion is performed on the lowest level items (items with no substructure).

EZECONVT is initialized to blanks. You must code your program to move a valid conversion table name to EZECONVT for conversion to occur. You can set EZECONVT to an asterisk to use the default conversion table for the national language specified in the EZERNLS environment variable.

If EZECONVT must be set to different values for different functions, code the program to move the correct value to EZECONVT immediately prior to processing the function that uses it.

Prior to its use, the value in EZECONVT is folded to uppercase. However, the value in the special function word EZECONVT remains unchanged. The special function word EZECONVT will test true when compared against the lowercase version if that is how it was initialized.

For additional information on cooperative processing and data conversion, refer to the sections on designing cooperative programs and converting data format in the *Design Guide* document.

Target environments for EZECONVT

Environment	Compatibility considerations
VM CMS	Not supported. EZECONVT has no effect because access to remote programs and files is not supported.
VM batch	Same as VM CMS.

Environment	Compatibility considerations
CICS for MVS/ESA	None.
MVS/TSO	Same as VM CMS.
MVS batch	Same as VM CMS.
IMS/VS	Same as VM CMS.
IMS BMP	Same as VM CMS.
CICS for VSE/ESA	None.
VSE batch	Same as VM CMS.
CICS for OS/2	None.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	None.
Windows Java (GUI)	None.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
CICS for AIX	None.
Solaris	None.
CICS for Solaris	None.
Windows NT (C++)	None.
Windows NT (Java)	None.
CICS for Windows NT	None.
Test Facility	None.

Example for EZECONVT

```
MOVE CNVTABLE TO EZECONVT;
```

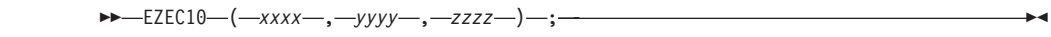
EZEC10

EZEC10 verifies a modulus-10 check-digit.

Uses

You can use EZEC10 as any of the following:
- The function name in a function invocation statement
- A map variable field edit routine

The following is the calling sequence for EZEC10:



xxxx
A character data item in working storage that contains the number for which you want to verify a check digit, including a position for the check digit.

yyyy
A binary data item of less than 5 digits that contains the number of characters to be used in item xxxx, including the check digit.

zzzz
A binary data item of less than 5 digits that returns a 0 if the number is a modulus-10 number, or a 1 if it is not.

Definition considerations for EZEC10

When used as a map variable field edit routine, the value is checked for the defined field length to insure that it passes the modulus-10 check. If the check fails, the program user is prompted to correctly enter the data.

Target environments for EZEC10

Supported in all environments without compatibility considerations.

Example for EZEC10

In the following example, `myinput` is defined as character data containing the value 1734284 (the rightmost 4 is the entered self-checking digit, not part of the base number), `mylength` is a binary data item containing the value 7, and `myresult` is a binary data item whose value will be set by the EZEC10 routine.

```
EZEC10(myinput,mylength,myresult);
```

EZEC10 derives the modulus-10 check digit, using the following algorithm:

1. Multiply the units position of the base number of source data by 2, and multiply every alternate position, moving right to left, by 2:

```
1    7    3    4    2    8    4 (Self-checking digit)
   x 2       x 2       x 2

  ____      ____      ____
   14         8        16
```

2. Add the `digits` of the products to the digits of the base number that were not multiplied by 2:

```
1 + 1 + 4 + 3 + 8 + 2 + 1 + 6 = 26
```

3. Subtract the sum from the next-highest number ending in 0 to get the check digit. (If the difference is 10, 0 is used.)

   ```
   30 - 26 = 4
   ```

   The resulting digit is the self-checking digit. The variable `myresult` is set to 0 if the self-checking digit is generated or 1 if not.

EZEC11

EZEC11 verifies a modulus-11 check-digit.

Uses

You can use EZEC11 as any of the following:
- The function name in a function invocation statement
- A map variable field edit routine

The following is the calling sequence for EZEC11:

xxxx
A character data item in working storage that contains the number for which you want to verify a check digit, including a position for the check digit.

yyyy
A binary data item of less than 5 digits that contains the number of characters to be used in item `xxxx`, including the check digit.

zzzz
A binary data item of less than 5 digits that returns a 0 if the number is a modulus-11 number, or a 1 if it is not.

Definition considerations for EZEC11

When used as a map variable field edit routine, the value is checked for the defined field length to insure that it passes the modulus-11 check. If the check fails, the program user is prompted to correctly enter the data.

Target environments for EZEC11

Supported in all environments without compatibility considerations.

Example for EZEC11

In the following example, `myinput` is defined as character data containing the value 56621865 (the rightmost 5 is the entered self-checking digit, not part of the base number), `mylength` is a binary data item containing the value 8, and `myresult` is a binary data item whose value will be set by the EZEC11 routine.

EZEC11

```
EZEC11(myinput,mylength,myresult);
```

EZEC11 derives the modulus-11 check digit using the following algorithm:

1. Multiply the units (rightmost) digit of the base number by 2, the tens position by 3, the hundreds position by 4, and so on, until 7 is used as a multiplier. If there are more digits to multiply, begin the sequence again using 2 as a multiplier:

```
  5    6    6    2    1    8    6    5 (Self-checking digit)
x 2  x 7  x 6  x 5  x 4  x 3  x 2
 10   42   36   10    4   24   12
```

2. Add the products of step 1:

```
10 + 42 + 36 + 10 + 4 + 24 + 12 = 138
```

3. Divide the sum of the products by 11:

```
      12
   ________
11 |138
    11
   ____
    28
    22
   ____
     6
```

   Subtract the remainder from 11 to get the self-checking digit. (If the remainder is 0 or 1, 0 is used.)

```
11 - 6 = 5
```

   The resulting digit is the self-checking digit. The variable `myresult` is set to 0 if the self-checking digit is generated or 1 if not.

EZEDAY

EZEDAY retrieves the current system date in Julian format (YYDDD). EZEDAY is automatically updated each time it is referenced by the program.

The Julian date is presented in a numeric format without separator characters.

The retrieved date format is valid for use in variable fields defined with a Julian date edit mask with a two-digit year.

Uses

You can use EZEDAY as the following:

- The source operand in a MOVE, MOVEA, or assignment statement

The receiver can be a map field or data item.

The characteristics of EZEDAY follow:

Data type
Numeric

Data length in bytes
5

Value saved across segments
No

Target environments for EZEDAY

Supported in all environments without compatibility considerations.

Example for EZEDAY

```
MOVE EZEDAY TO MYDAY;
```

EZEDAYL

EZEDAYL retrieves the current date in Julian format (YYYYDDD). EZEDAYL is automatically updated each time it is referenced by the program.

The Julian date is presented in a numeric format without separator characters.

The retrieved date format is valid for use in variable fields defined with a Julian date edit mask with a four-digit year.

Uses

You can use EZEDAYL as the following:
- The source operand in a MOVE, MOVEA, or assignment statement

The receiver can be a map field or data item.

The characteristics of EZEDAYL follow:

Data type
Numeric

Data length in bytes
7

Value saved across segments
No

Target environments for EZEDAYL

Supported in all environments without compatibility considerations.

Example for EZEDAYL

```
MOVE EZEDAYL TO DATE_ITEM;
```

EZEDAYLC

EZEDAYLC retrieves the current date in the system default long Julian format.

The system default format for the Julian date includes separator characters. The environment variable EZERJULL_xxx where xxx determines the Julian format for dates.

The xxx specifies the language code. For example, the following are applicable language codes:

CHS Simplified Chinese

PTB Brazilian Portuguese

ENU English

JPN Japanese

KOR Korean

The retrieved date format is valid for use in variable fields defined with a Julian date edit mask with a four-digit year.

For OS/2, if EZERJULL_xxx does not exist, the default Julian format is derived from the OS/2 system settings.

For 370 environments, if EZERJULL_xxx does not exist, the default Julian format is specified during installation.

Uses

You can use EZEDAYLC as the following:
- The source operand in a MOVE, MOVEA or assignment statement

The receiver can be a map field or a data item.

The characteristics of EZEDAYLC follow:

Data type
Character

Data length in bytes
8

Value saved across segments
No

Target environments for EZEDAYLC

Supported in all environments without compatibility considerations.

Example for EZEDAYLC

```
MOVE EZEDAYLC TO DAY_ITEM;
```

EZEDEST

EZEDEST dynamically changes the system resource name associated with a record while the program is running.

You can change the physical file or data set associated with the logical file name defined for a record by moving a data item or literal containing the new system resource name for the file into the special function word EZEDEST. This change takes place dynamically while the program is running.

EZEDEST must be a qualified record name (recordname.EZEDEST) unless it is implicitly qualified as in the following conditions:

- Only one record is used as an I/O object in the program.
- EZEDEST is used in a function that has a record as the I/O object. The record name implicitly qualifies EZEDEST.
- Multiple records are used as I/O objects in the program, but all records have the same file name. The first record that appears as an I/O object is used as the implicit qualifier.

Uses

You can use EZEDEST as any of the following:

- The source operand of a MOVE, MOVEA, or assignment statement
- The target operand of an assignment or MOVE statement
- Data item 1 and 2 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand on the TEST statement

The characteristics of EZEDEST follow:

Data type
Character

Data length in bytes
Varies by file type

Value saved across segments
Yes

Definition considerations for EZEDEST

You must ensure that the value moved into EZEDEST is a valid system resource name for the run-time environment and file type specified when the program was generated. The Target environments section describes the valid name formats by environment and file type.

If more than one record has the same file name specified, modification of EZEDEST for any record with that file name changes the setting of EZEDEST for all records in the program with the same file name.

Whenever an I/O operation is performed for a record, the program performs the I/O on the physical file or data set whose name is in the EZEDEST item for the record.

Use a CLOSE I/O option to close the file associated with the current setting of EZEDEST. When a run unit ends or a segmented CONVERSE occurs, all open files are closed, except when running with the test facility.

The previously opened physical file closes when a I/O option for a record with the same VisualAge Generator file name is run and EZEDEST has been modified. If two programs are using the same VisualAge Generator file name, you must ensure EZEDEST contains the same value. Otherwise, the previously opened physical file is closed and the new one is opened.

Prior to its use, the value in EZEDEST is folded to uppercase. However, the value in the special function word EZEDEST remains unchanged.

The special function word EZEDEST tests true when compared against the lowercase version if that is how it was initialized.

EZEDEST is initialized to the system resource name specified during generation.

Using EZEDEST with Files Shared across Programs

If EZEDEST is used, each program that accesses the file must set EZEDEST for the file. If two programs in the same run unit access the same logical file, each program must set EZEDEST to the same system resource name to ensure that both programs access the same physical file at run time.

Using EZEDEST with message queue records

If you are using VAGen support for MQSeries message queues, the program can dynamically change the message queue associated with the record by moving a data item or literal containing the system resource name for the message queue record into the special function word EZEDEST.

The system resource name for message queue records defines the queue manager name and queue name. Specify the name using the format:

```
queue_manager_name:queue_name
```

where the names are separated by a single colon (:) or specify the queue_name by itself if you omit the queue_manager_name. The system

resource name is used as the initial value for the EZEDEST item for the message queue record and identifies the default queue associated with the record.

VAGen uses the system resource name on ADD and SCAN I/O operations for the message queue record. The queue name identifies the queue that is accessed by the operation. The queue manager name identifies the queue manager on which the queue is defined. The default queue manager is the queue manager to which the program is connected.

If there is not already an active connection, VAGen uses the queue manager name to connect to the queue manager before accessing the queue. If no queue manager name is specified, VAGen connects to the default queue manager for the system.

If the system resource name is not specified in a resource association file, a default system resource name is defined by the **File name** property of the message queue record.

Specifying System Resource Name at Generation

If two programs in the same run unit access the same logical file, you must specify the same system resource name for the logical file at generation to ensure that both programs access the same physical file at run time.

Target environments for EZEDEST

Environment	Compatibility considerations
VM CMS	**SEQ** Not supported.
	SEQRS The value is a CMS file name (fn ft fm), a fully qualified name of an MVS file on an OS formatted mini disk (for input only), or an 8-byte DD name for a system sequential file associated with a serial file. The DD name is the value from a previously specified FILEDEF or DLBL command. If no value has been set for EZEDEST, the program first looks to see if a data set has been preallocated using the logical file name as the DD name. Otherwise, the system resource name specified at generation is used to access the file. When you move a data set name into EZEDEST, the resource is dynamically allocated to the EZEDEST record file name using a CMS FILEDEF command. The allocation is done with the DISP option set to OLD. This will force the data set to be rewritten each time it is opened for output.
	VSAM Not supported.
	VSAMRS The value is a 44-byte data set name or an 8-byte DD name for a VSAM file associated with an indexed, relative, or serial file. The DD name is the value from a previously specified DLBL command. If no value has been set for EZEDEST for a file, the program first looks to see if a data set has been preallocated using the logical file name as the DD name. Otherwise, the system resource name specified at generation is used to access the file. When you move a data set name into EZEDEST, the resource is dynamically allocated to the EZEDEST record file name using a CMS DLBL command. The allocation is done with the DISP option set to OLD, which forces the data set to be written each time it is open for output.

Environment	Compatibility considerations
VM batch	**GSAM** Not supported. **SEQ** Not supported. **SEQRS** Same as VM CMS. **VSAM** Not supported. **VSAMRS** Same as VM CMS.

Environment	Compatibility considerations
CICS for MVS/ESA	**SPOOL** The value is the input or output file name for a JES SPOOL file associated with a serial file. **Input file name:** `userid.class`. The `userid` parameter is a 4- to 8-character external writer name or an asterisk (*). If an external writer name is used, CICS requires that the first 4 characters of the external writer name be the same as the first 4 characters of the CICS APPLID used to identify the CICS region to ACF/VTAM. The `class` parameter is a 1-character spool class. Class is optional and defaults to "A". The maximum name size is 10 bytes. Refer to the CICS customization manual for more information. **Output file name:** `nodeid.userid.class`. The `nodeid` parameter is either a 1- to 8-character system node ID, or an asterisk (*). The `userid` parameter is a 1- to 8-character system user ID, or an asterisk (*). The `class` parameter is a 1-character spool class. `Class` is optional and defaults to "A". If `class` is not specified, `userid` is also optional and defaults to the CICS user ID (the same value stored in EZEUSRID). The maximum name size is 19 bytes. Refer to the CICS customization manual for more information. **TEMPMAIN** The value is an 8-byte queue name for a main temporary storage queue associated with a relative or serial file. Some queue names are reserved for use by VisualAge Generator Server for MVS, VSE, and VM and are prefixed with EZE. When a temporary storage queue name is moved into EZEDEST, the queue is dynamically created, unless it already exists. A single temporary storage queue file cannot be shared by multiple users at the same time. A CICS ENQ is used to serialize access to the file the first time it is referenced in the program. The DEQ is issued when the program closes the file. **TEMPAUX** TEMPAUX is like TEMPMAIN, except that it is for an auxiliary temporary storage queue. **TRANSIENT** The value is a 4-byte DCT name for a transient data queue associated with a serial file. The transient data queue name must be defined to CICS. **VSAM** The value is an 8-byte FCT name for a VSAM file associated with an indexed, relative, or serial file.

Environment	Compatibility considerations
MVS/TSO	**SEQ** Not supported. **SEQRS** The value is a 54-byte data set name or an 8-byte DD name for a system sequential file associated with a serial file. If a value has not been set for EZEDEST for a file, the program first looks to see if a data set has been preallocated using the logical file name as the DD name. Otherwise, the system resource name specified at generation is used to access the file. When you move a data set name into EZEDEST, the resource is dynamically allocated to the EZEDEST record file name using an MVS supervisor call (SVC99). The allocation is done with the DISP option set to SHR. This will force the data set to be rewritten each time it is opened for output. **VSAM** Not supported. **VSAMRS** The value is a 44-byte data set name or an 8-byte DD name for a VSAM file associated with an indexed, relative, or serial file. If no value has been set for EZEDEST for a file, the program first looks to see if a data set has been preallocated using the logical file name as the DD name. Otherwise, the system resource name specified at generation is used to access the file. When you move a data set name into EZEDEST, the resource is dynamically allocated to the EZEDEST record file name using an MVS supervisor call (SVC99). The allocation is done with the DISP option set to SHR, which forces the data set to be written each time it is open for output.
MVS batch	**GSAM** Not supported. **SEQ** Not supported. **SEQRS** Same as SEQRS in MVS/TSO. **VSAM** Not supported. **VSAMRS** Same as VSAMRS in MVS/TSO.

Environment	Compatibility considerations
IMS/VS	**MMSGQ** The value in EZEDEST is not folded to uppercase for this file type. **Input file name:** Not supported. **Output file name:** The value is the 8-byte logical terminal name or transaction code for a multisegment message queue associated with a serial file. The file must be associated with a modifiable alternate or modifiable express alternate PCB. The transaction code or terminal name must be defined to the IMS system. **SMSGQ** The value in EZEDEST is not folded to uppercase for this file type. **Input file name:** Not supported. **Output file name:** The value is the 8-byte logical terminal name or transaction code for a single-segment message queue associated with a serial file. The file must be associated with a modifiable alternate or modifiable express alternate PCB. The transaction code or terminal name must be defined to the IMS system.
IMS BMP	**GSAM** Not supported. **MMSGQ** Same as MMSGQ in IMS/VS. **SEQ** Not supported. **SEQRS** Same as SEQRS in MVS/TSO. **SMSGQ** Same as SMSGQ in IMS/VS. **VSAM** Not supported. **VSAMRS** Same as VSAMRS in MVS/TSO.

Environment	Compatibility considerations
CICS for VSE/ESA	**SPOOL** The value is the input or output system resource name for a SPOOL file. No error checking is done by VisualAge Generator Server for MVS, VSE, and VM to ensure that a correct combination of values is specified for the qualifiers of the system resource name. (For example, `node` must be the current system node when `queue` is RDR; it is not valid to send a VSE/POWER RDR file to another node.) Instead, the values for each of the system resource name qualifiers will be sent to VSE/POWER. The return code from VSE/POWER will be placed in the special function word EZERT8, where it can be accessed by the VisualAge Generator program. **Input file name:** `userid.class`. `Userid` is a 4- to 8-character external writer name or an asterisk (*). If an external writer name is used, CICS requires that the first 4 characters be the same as the first 4 characters of the CICS APPLID used to identify the CICS region to ACF/VTAM. `Class` is a 1-character spool class. `Class` is optional and defaults to “A”.

Environment	Compatibility considerations
CICS for VSE/ESA (continued)	**SPOOL** The output system resource name format for a spool file is: `jobname.queue.disp.form.node.userid.parm`. `Jobname` is 1- to 8-character name that defines the jobname for the VSE/POWER queue part. `Jobname` is used except when `queue` is PUN or LST. In which case the value in `jobname` is ignored, and the VSE/POWER queue part `jobname` is the CICS for VSE/ESA program ID. For all other cases, an asterisk (*) for this qualifier defaults to the VisualAge Generator file name for the record. You must specify a `jobname` qualifier or an asterisk (*). All other qualifiers can contain an asterisk or be blank. If a qualifier is blank, you cannot specify any subsequent qualifiers. `Queue` is 3 characters that identify the destination VSE/POWER queue for the file. The CICS Report Control Facility (RCF) is used for files that specify RDR or PRT in this field. • RDR for job output • LST for list output • PUN for punch output • PRT for list output (using CICS Report Control Facility) Any other characters for `queue` cause a spool name error. An asterisk (*) or a blank for this qualifier defaults to the PRT queue. LST or PRT specifies that the file is to be a part of the VSE/POWER LST queue, but PRT uses RCF commands while LST does not. If you try to use RCF when you do not have RCF installed on your CICS system, CICS returns an error. This error might be an AEY9 transaction abend, a NO SPOOL condition, or the message: `Spooling system is not available.` When `queue` is PRT or LST, the file is opened by VisualAge Generator Server for MVS, VSE, and VM with the ASA option. This option specifies that the report is created using an American National Standard printer-control character at the beginning of each line of data. If you are using a serial file, ensure that valid carriage control characters are used. If the file is a print file, then VisualAge Generator Server for MVS, VSE, and VM will add the American National Standard printer-control character for you.

Environment	Compatibility considerations
CICS for VSE/ESA (continued)	**SPOOL** The output system resource name format for a spool file is: `jobname.queue.disp.form.node.userid.parm`. `Disp` is a single character that specifies the VSE/POWER disposition status of the queue part after it closes: **D** Process the job and delete it after processing **H** Hold in queue until released **K** Process the job and keep it in the queue after processing **L** Leave in queue until released Any other characters cause a spool name error. This qualifier is not applicable when `queue` is LST or PUN. An asterisk or blank for this qualifier defaults to "D". `Form` is 4 characters that identify the print output. An asterisk or a blank defaults to your location's standard form. This qualifier is applicable when `queue` is LST or PRT and is ignored for all other queues. `Node` is 1 to 8 characters that specify the system node identifier. An asterisk or a blank defaults to the current system node identifier. `Userid` is a 4- to 8-character external writer name or an asterisk (*). If an external writer name is used, CICS requires that the first 4 characters be the same as the first 4 characters of the CICS APPLID used to identify the CICS region to ACF/VTAM. `Parm` is valid when queue is LST and is ignored on all other queues. `Parm` specifies output operands for files on the VSE/POWER LST queue, which are used as input in the OUTDESCR option of the VSECICS SPOOLOPEN OUTPUT command.

Environment	Compatibility considerations

CICS for VSE/ESA (continued)	**SPOOL**
	You must specify this qualifier in the correct format for the OUTDESCR option. The qualifiers use the same keywords and values that are used on the VSE/POWER LST statement for user-defined output operands, but the syntax varies slightly. For example, if you want to use FORMDEF FORM1 and PAGEDEF PAGE1, the qualifier string would be:
	`FORMDEF(FORM1) PAGEDEF(PAGE1)`
	And the spool file might look like this:
	`JOBNAME1.LST.*.*.*.*.FORMDEF(FORM1) PAGEDEF(PAGE1)`
	The length of the qualifier string is variable and depends on the length of the spool file specification up to this point. The total length of the spool file specification cannot exceed 65 characters.
	TEMPMAIN
	Same as CICS for MVS/ESA.
	TEMPAUX
	Same as CICS for MVS/ESA.
	TRANSIENT
	Same as CICS for MVS/ESA.
	VSAM The value is a 7-byte FCT name for a VSAM file associated with an indexed, relative, or serial file.

Environment	Compatibility considerations
VSE batch	**SEQ** Not supported. **SPOOL** The system resource name format for a SPOOL output file is: `jobname.queue.class.disp.form.node.userid` **jobname** Same as CICS for VSE/ESA. **queue** An asterisk (*) or a blank in this field defaults to the LST queue. PRT can be used, but the queue parameter will be changed to LST by VisualAge Generator Server for MVS, VSE, and VM. **class** The class parameter is a 1-character spool class. Class is optional and defaults to “A”. **disp** Same as CICS for VSE/ESA. **form** Same as CICS for VSE/ESA. **node** Same as CICS for VSE/ESA. **userid** Same as CICS for VSE/ESA. **VSAM** Not supported. **VSAMRS** The value is a 7-byte DD name for a VSAM file associated with an indexed, relative, or serial file. If no value has been set for EZEDEST for a file, the program first looks to see if a data set has been preallocated using the logical file name as the DD name. Otherwise, the system resource name specified at generation is used to access the file. Dynamic allocation of resources to EZEDEST is not supported. Any dynamic allocation attempts will result in message ELA0007P.

Environment	Compatibility considerations
CICS for OS/2	**OS2COBOL** The value is a 65-byte OS/2 file name for a native COBOL data file associated with an indexed, relative, or serial file. File sharing for COBOL-managed data files is not supported. Whenever the file is opened, an exclusive lock is obtained on the file until it is closed. **TEMPMAIN** Same as CICS for MVS/ESA. **TEMPAUX** Same as CICS for MVS/ESA. **TRANSIENT** Same as CICS for MVS/ESA. **VSAM** Same as CICS for MVS/ESA.
OS/400	The filetype must be **SEQ** or **VSAM**. The value can be moved to EZEDEST in one of the following ways: **LIB/FILE MEMBER** Explicitly specify Library, File and Member **LIB/FILE** The first member in the file will be used **FILE MEMBER** *LIBL will be used to find the file **FILE** *LIBL will be used to find the file and the first member in that file will be used. The OVRDBF command is used to support EZEDEST on OS/400. If the value in EZEDEST is modified, the following is done while performing a File I/O: 1. CLOSE old file 2. Override to new file name in EZEDEST 3. OPEN new file The value set in EZEDEST is propagated from the call level and changed to all its subordinate call levels. However, it is not propagated if the file has been previously opened by that program.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	The file name length is dependent upon the system.
AIX	The file name length is dependent upon the system.
HP-UX	The file name length is dependent upon the system.

Environment	Compatibility considerations
Solaris	The file name length is dependent upon the system.
CICS for Solaris	**SEQ** The value is the Solaris file name. On Solaris, the maximum length can vary. **TEMPMAIN** The value is a 1- to 8-byte temporary storage queue name. When a temporary storage queue name is moved into EZEDEST, the queue is dynamically created, unless it already exists. A single temporary storage queue file cannot be shared by multiple users at the same time. A CICS ENQ is used to serialize access to the file the first time it is referenced in the program. The DEQ is issued when the program closes the file. **TEMPAUX** TEMPAUX is like TEMPMAIN, except that it is for an auxiliary temporary storage queue. **TRANSIENT** The value is a 1- to 4-byte transient data queue name as defined in the CICS destination control table (DCT). **VSAM** Same as VSAM for CICS for MVS/ESA.
CICS for AIX	**SEQ** The value is the AIX file name. On AIX, the maximum length can vary. **TEMPMAIN** The value is a 1- to 8-byte temporary storage queue name. When a temporary storage queue name is moved into EZEDEST, the queue is dynamically created, unless it already exists. A single temporary storage queue file cannot be shared by multiple users at the same time. A CICS ENQ is used to serialize access to the file the first time it is referenced in the program. The DEQ is issued when the program closes the file. **TEMPAUX** TEMPAUX is like TEMPMAIN, except that it is for an auxiliary temporary storage queue. **TRANSIENT** The value is a 1- to 4-byte transient data queue name as defined in the CICS destination control table (DCT). **VSAM** Same as VSAM for CICS for MVS/ESA.

Environment	Compatibility considerations
Windows NT (C++)	Same as OS/2 (C++).
Windows NT (Java)	None.
CICS for Windows NT	Same as CICS for AIX.
Test Facility	When running a program, the same system resource name should not be used for a program being run in the test facility as one being executed as a generated program.

Example for EZEDEST

```
MOVE IITEM TO MYREC.EZEDEST;

MOVE 'IMSTRNX' TO MYREC1.EZEDEST;

IF EZESYS IS MVSCICS;
   MYREC1.EZEDEST = 'TDQ1';
ELSE;
   IF EZESYS IS TSO;
      MYREC1.EZEDEST = 'MYUSERID.TEST.RECFILE';
   END;
END;
```

EZEDESTP

EZEDESTP changes the name of the printer while the program is running. You can change the name of the destination associated with the print file by moving a data item or literal containing the new printer destination to the special function word EZEDESTP. For some file types, multiple printers can be open simultaneously.

Uses

You can use EZEDESTP as any of the following:

- The source operand of a MOVE, MOVEA, or assignment statement
- The target operand of a MOVE or an assignment statement
- Data item 1 and 2 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand on the TEST statement

The characteristics of EZEDESTP follow:

Data type

Character

Data length in bytes
Varies by file type

Value saved across segments
Yes

Definition considerations for EZEDESTP

To prevent interleaving of multiple print outputs, the print file is kept open until one of the following occurs:

- A CLOSE I/O option is performed against the map
- A segmentation break occurs (except in the test facility)
- The main program ends.
- A called program ends and was called by a non-VisualAge Generator program

The CLOSE I/O option closes the file only for the current value of EZEDESTP. A printer is not closed just because EZEDESTP has been used to change the physical file to which a printout is being directed. For the file types where multiple print files cannot be open, the printer will be closed when EZEDESTP is used to change the print destination prior to execution of the next DISPLAY of a print map.

You must ensure that the value moved into EZEDESTP is a valid system resource name for the run-time environment and print file type specified when the program was generated. The target environments section describes valid name formats by environment and file type.

The value of EZEDESTP is unique for each program. If a program passes control to another program, the value of EZEDESTP is reset to its default value.

EZEDESTP is initialized to the system resource name specified during generation or test execution.

Prior to its use, the value in EZEDESTP is folded to uppercase. However, the value in the special function word EZEDESTP remains unchanged.

The special function word EZEDESTP will test true when compared against the lowercase version if that is how it was initialized.

Target environments for EZEDESTP

Environment	Compatibility considerations	
VM CMS	**SEQ**	Not supported.
	SEQRS	The value is a CMS file name (fn ft fm), or an 8-byte DD name for a system sequential file associated with a print file. The DD name is the value from a previously specified FILEDEF or DLBL command. If no value has been set for EZEDESTP, the program first looks to see if a data set has been preallocated using the logical file name as the DD name. Otherwise, the system resource name specified at generation is used to access the file. When you move a data set name into EZEDESTP, the resource is dynamically allocated to EZEPRINT using a CMS FILEDEF command. This forces the data set to be rewritten each time it is opened for output. The allocation is done with the DISP option set to OLD. Multiple files can be open simultaneously.
VM batch	**GSAM**	Not supported.
	SEQ	Not supported.
	SEQRS	Same as VM CMS.

Environment	Compatibility considerations
CICS for MVS/ESA	If generation option /PRINTDEST is TERMID, and the program was started with a CREATX statement that had the recip qualifier set to binary zeros and specified a prid qualifier, then EZEDESTP is initialized to the value specified for the prid qualifier. If generation option PRINTDEST is EZEP, then EZEDESTP is initialized to the value associated with EZEPRINT at generation. For any batch program that was not started by CREATX, EZEDESTP defaults to the CICS EIBTRMID. **SPOOL** The value is the output file name for a JES SPOOL file associated with a serial file. **Output file name:** `nodeid.userid.class` The nodeid qualifier is either a 1 to 8-character system node ID, or an asterisk (*). The userid qualifier is either a 1 to 8-character system user ID, or an asterisk (*). The class qualifier is a 1-character spool class. Class is optional and defaults to 'A'. If a class is not specified, userid is also optional and defaults to the CICS user ID (the same value stored in EZEUSRID). The maximum name size is 19 bytes. Refer to the CICS customization manual for more information. Multiple files can be open simultaneously. **TRANSIENT** The value is a 4-byte DCT name for a transient data queue associated with a serial file. The transient data queue name must be defined to CICS. Multiple files can be opened simultaneously.

Environment	Compatibility considerations
MVS/TSO	**SEQ** Not supported. **SEQRS** The value is a 54-byte data set name or an 8-byte DD name for a system sequential file associated with a print file. Multiple files can be open simultaneously. If EZEDESTP has not been set by the program and the file type is SEQRS, the program first looks for a file allocated to DD name EZEPRINT. In all other cases, the system resource name specified for EZEPRINT at generation is used as the initial value if EZEDESTP is not explicitly set by the program. When you move a data set name into EZEDESTP for an SEQRS file, the file is dynamically connected using SVC99 dynamic allocation. When dynamic allocation is performed, the DISP option on the SVC99 is set to SHR. This forces the data set to be rewritten each time it is opened for output.
MVS batch	**GSAM** Not supported. **SEQ** Not supported. **SEQRS** Same as SEQRS in MVS/TSO. Multiple files can be open simultaneously.
IMS/VS	**SMSGQ** The value is an 8-byte logical terminal name or transaction code for a single-segment message queue associated with a print file. The file must be associated with a modifiable alternate or modifiable express alternate PCB. The terminal name must be defined to the IMS system. Multiple files cannot be open simultaneously. The value in EZEDESTP is not folded to uppercase for this file type.
IMS BMP	**GSAM** Not supported. **SEQ** Not supported. **SEQRS** Same as MVS/TSO. **SMSGQ** Same as IMS/VS.
CICS for VSE/ESA	**SPOOL** Same CICS for VSE/ESA SPOOL for EZEDEST. **TRANSIENT** Same as CICS for MVS/ESA.

Environment	Compatibility considerations

VSE batch	**SEQ SPOOL** Not supported.
	
	The system resource name format for a SPOOL output file is:
	`jobname.queue.class.disp.form.node.userid`
	
	or
	`jobname.queue.class.disp.form.node.userid.fcb.copy`
	
	jobname Same as CICS for VSE/ESA.
	
	queue An asterisk (*) or blank in this field defaults to the LST queue. PRT can be used but the queue parameter will be changed to LST by Server for MVS, VSE, and VM.
	
	class The class parameter is a 1-character spool class. Class is optional and defaults to “A”.
	
	disp Same as CICS for VSE/ESA.
	
	form Same as CICS for VSE/ESA.
	
	node Same as CICS for VSE/ESA.
	
	userid Same as CICS for VSE/ESA.
	
	fcb 1-to 8-character name that specifies the FCB-image phase which VSE/POWER is to use for printing the related job output. The named phase must be cataloged in a sublibrary defined as accessible from the VSE/POWER partition.
	
	copy The number of the copies to be printed from LST queue. The valid number is from 0 to 255.
	
	Both FCB and COPY parameters are positional and optional. Use an asterisk (*) for defaults. The default FCB phase name is the one setup at IPL time and the default COPY is 1.

Environment	Compatibility considerations
CICS for OS/2	**OS2COBOL** The value of a 65-byte OS/2 file name for a native COBOL data file associated with a print file. File sharing for COBOL-managed data files is not supported. Whenever the file is opened, an exclusive lock is obtained on the file until it is closed. **TRANSIENT** Not supported.
OS/400	The filetype must be **SEQ**. The value can be moved to EZEDESTP using `FILE - *LIBL` to find the file. The OVRPRTF command is used to support EZEDESTP on OS/400. If the value in EZEDESTP is modified, the following is done at the time of a DISPLAY I/O option: 1. CLOSE old printer file 2. Override to new printer file name in EZEDESTP (OVRPRTF) 3. OPEN new printer file The value set in EZEDESTP is propagated from the call level and changed to all its subordinate call levels. However, it is not propagated if the file has been previously opened by that program.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	The file name length is dependent upon the system.
AIX	The file name length is dependent upon the system. In AIX, the maximum length can vary.
HP-UX	The file name length is dependent upon the system. In HP/UX, the maximum length can vary.
CICS for AIX	The EZEDESTP value is a transient data queue name. The default is EZEP. Refer to the *VisualAge Generator Server Guide for Workstation Platforms* document for information on how to define the queue and associate it with a transaction that writes queue contents to a terminal printer.
Solaris	The file name length is dependent upon the system. In AIX, the maximum length can vary.

Environment	Compatibility considerations
CICS for Solaris	The EZEDESTP value is a transient data queue name. The default is EZEP. Refer to the *VisualAge Generator Server Guide for Workstation Platforms* document for information on how to define the queue and associate it with a transaction that writes queue contents to a terminal printer.
Windows NT (C++)	Same as OS/2 (C++).
Windows NT (Java)	Not supported.
CICS for Windows NT	Same as CICS for AIX.
Test Facility	A print file is not closed on a segmentation break. The same system resource name should not be used for a program being run in the test facility as for one being run as a generated COBOL program.

Examples for EZEDESTP

```
MOVE ITEM TO EZEDESTP;

MOVE 'ACCTRPT' TO EZEDESTP;

MOVE 'QUAL.PDSFILE(PART)' to EZEDESTP;
```

EZEDLCER (DL/I)

EZEDLCER contains the CICS for MVS/ESA and CICS for VSE/ESA error code for a DL/I call issued for a DL/I function.

Uses

You can use EZEDLCER as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand in a TEST statement

The characteristics of EZEDLCER follow:

Data type
Character

Data length in bytes
2

Value saved across segments
No

Definition considerations for EZEDLCER

EZEDLCER is reset each time a DL/I call is issued. You can code your program to check EZEDLCER in the function error routine if EZEDLERR or EZEFEC is set to continue after hard errors. EZEDLCER is read-only and cannot be reset by the program.

For more information about return codes, refer to the CICS application's reference for your version of CICS.

Target environments for EZEDLCER

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	The value of EZEDLCER is always 00.
MVS batch	Same as MVS/TSO.
IMS/VS	Same as MVS/TSO.
IMS BMP	Same as MVS/TSO.
CICS for VSE/ESA	None.
VSE batch	Same as MVS/TSO.
CICS for OS/2	Same as MVS/TSO.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
CICS for AIX	Not supported.

Environment	Compatibility considerations
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Test Facility	If the DL/I execution environment is CICS, and the program is handling hard errors (EZEFEC=1), the EZEDLCER return field is set with the CICS return codes that correspond to the results returned from the DDBA API. Otherwise, if the schedule function fails, the test facility returns an error.

Example for EZEDLCER

Use an IF statement to test the status information. In the following example, the statement tests true if the CICS return code is not normal.

```
IF EZEDLCER NE '00';
END;
```

EZEDLCON (DL/I)

EZEDLCON contains the condition code returned by CICS for MVS/ESA and CICS for VSE/ESA for a DL/I call issued for a DL/I function.

EZEDLCON can contain the following values:

00 Normal response
08 Request was not valid
0C Not open

Uses

You can use EZEDLCON as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand in a TEST statement

The characteristics of EZEDLCON follow:

Data type
Character

Data length in bytes
2

Value saved across segments
No

Definition considerations for EZEDLCON

EZEDLCON is reset each time a DL/I call is issued. You can code your program to check EZEDLCON in the function error routine if EZEFEC or EZEDLERR is set to continue after hard errors.

EZEDLCON is read-only; it cannot be reset by the program.

For more information about return codes, refer to the CICS application programmer's reference for your version of CICS.

Target environments for EZEDLCON

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	The value of EZEDLCON is always 00.
MVS batch	Same as MVS/TSO.
IMS/VS	Same as MVS/TSO.
IMS BMP	Same as MVS/TSO.
CICS for VSE/ESA	None.
VSE batch	Same as MVS/TSO.
CICS for OS/2	Same as MVS/TSO.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
CICS for AIX	Not supported.

Environment	Compatibility considerations
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Test Facility	If the DL/I execution environment is CICS, and the program is handling hard errors (EZEFEC=1), the EZEDLCON return field is set with the CICS return codes that correspond to the results returned from the DDBA API. Otherwise, if the schedule function fails, the test facility returns an error.

Example for EZEDLCON

Use an IF statement to test the status information. In the following example, the statement tests true if the CICS error returned is not normal.

```
IF EZEDLCON NE '00';
END;
```

EZEDLDBD (DL/I)

EZEDLDBD contains the name of the database accessed by the last DL/I I/O function. The name is from the database PCB used by the DL/I call for the function.

Uses

You can use EZEDLDBD as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand in a TEST statement

The characteristics of EZEDLDBD follow:

Data type
Character

Data length in bytes
8

Value saved across segments
No

Definition considerations for EZEDLDBD

EZEDLDBD is set to blanks under the following conditions:

- Initially
- When the PSB is terminated
- When another program is called

EZEDLDBD is reset each time a DL/I call is issued. EZEDLDBD can be displayed to assist in error determination or tested in the function error routine to determine the outcome of a DL/I call.

EZEDLDBD is read-only; it cannot be reset by the program.

EZEDLDBD is not set following CSPTDLI service calls, or following DL/I calls to IMS message queues or GSAM files.

Target environments for EZEDLDBD

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	Not supported.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Java (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.

Environment	Compatibility considerations
CICS for AIX	Not supported.
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Test Facility	None.

Example for EZEDLDBD

```
IF EZEDLDBD NOT BLANKS;
END;
```

EZEDLERR (DL/I)

EZEDLERR controls processing for error conditions for I/O functions that have DL/I segments as I/O objects.

If EZEDLERR and EZEFEC are set to 0, or if an error routine is not specified, the program ends when a hard error occurs on a DL/I call.

If EZEDLERR or EZEFEC is set to 1, and an error routine is specified, the program does not end when a hard DL/I I/O error occurs. The program must handle hard errors by checking the DL/I EZE words that contain DL/I status information.

The default value of EZEDLERR is 0.

Uses

You can use EZEDLERR as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand in a MOVE or assignment statement
- An occurrence operand in a MOVEA statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

The characteristics of EZEDLERR follow:

Data type
Numeric

Data length in bytes
1

Value saved across segments
Yes

Definition considerations for EZEDLERR

Hard errors on DL/I calls occur under the following conditions:

- A CICS condition code (EZEDLCON) or error code (EZEDLCER) has a value other than 00
- A DL/I status code has a value other than one of the following:
  GA
  GB
  GD
  GE
  GK
  II
  AL

EZEDLERR has no effect on error processing following CSPTDLI service calls or DL/I calls to IMS message queues or GSAM files.

Target environments for EZEDLERR

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	Not supported.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	Not supported.

Environment	Compatibility considerations
AIX	Not supported.
HP-UX	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
CICS for AIX	Not supported.
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Test Facility	None.

Example for EZEDLERR

```
MOVE 1 TO EZEDLERR;
```

EZEDLKEY (DL/I)

EZEDLKEY contains the concatenated key of the lowest-level segment found by the last DL/I I/O function. The key is from the database PCB used by the DL/I call for the function.

Uses

You can use EZEDLKEY as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand on a TEST statement

The characteristics of EZEDLKEY follow:

Data type
Character

Data length in bytes
Variable

Value saved across segments
No

Definition considerations for EZEDLKEY

EZEDLKEY is set to blanks under the following conditions:

- Initially
- When the PSB is terminated
- When another program is called

EZEDLKEY is reset to the concatenated key each time a DL/I call is issued. It can be displayed to assist in error determination, or tested in the function error routine to determine the outcome of a DL/I call.

EZEDLKEY is read-only; it cannot be reset by the program.

EZEDLKEY is not set following CSPTDLI service calls or DL/I calls to IMS message queues or GSAM files.

Special function word EZEDLKYL contains the length of EZEDLKEY. If a program moves the contents of EZEDLKEY to a data item, the length value in EZEDLKYL is used to perform the move.

Target environments for EZEDLKEY

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	Not supported.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	Not supported.

Environment	Compatibility considerations
AIX	Not supported.
HP-UX	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
CICS for AIX	Not supported.
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Test Facility	The test facility requires the EBCDIC version of the Micro Focus Mainframe Express with IMS Option and Micro Focus Mainframe Express with IMS Option. The Micro Focus Mainframe Express with IMS Option emulator stores data internally in EBCDIC. The test facility converts the data to ASCII for display and use within the test facility. Data defined in the test facility is converted to EBCDIC before being put in the database. Following a DL/I I/O option, the information in the EZE DL/I status words is converted. EZEDLKEY is converted based on the key items defined for the records referenced by the call. If one of the referenced segments has no key item defined, conversion of the Key Feedback area will stop with that field.

Example for EZEDLKEY

```
MOVE EZEDLKEY TO ITEM1;
```

EZEDLKYL (DL/I)

EZEDLKYL contains the length of the concatenated key returned in special function word EZEDLKEY for the last DL/I I/O function. The length is from the database PCB used by the DL/I call for the function.

Uses

You can use EZEDLKYL as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- An occurrence operand in a MOVEA statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

The characteristics of EZEDLKYL follow:

Data type
Binary

Data length in bytes
4

Value saved across segments
No

Definition considerations for EZEDLKYL

EZEDLKYL is set to zero under the following conditions:
- Initially
- When the PSB is terminated
- When another program is called

EZEDLKYL is reset to the current key length each time a DL/I call is issued. It can be displayed to assist in error determination, or tested in the function error routine to determine the outcome of a DL/I call.

EZEDLKYL is read-only; it cannot be reset by the program.

EZEDLKYL is not set following CSPTDLI service calls or DL/I calls to IMS message queues or GSAM files.

Target environments for EZEDLKYL

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	Not supported.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.

Environment	Compatibility considerations
Windows Java (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
CICS for AIX	Not supported.
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Test Facility	None.

Example for EZEDLKYL

```
IF EZEDLKYL = 10;
END;
```

EZEDLLEV (DL/I)

EZEDLLEV contains the level number of the lowest-level segment found by DL/I in the last DL/I I/O function. The level number is from the database PCB used by the DL/I call for the function.

Uses

You can use EZEDLLEV as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- An occurrence operand in a MOVEA statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

The characteristics of EZEDLLEV follow:

Data type
Numeric

Data length in bytes
2

Value saved across segments
No

Definition considerations for EZEDLLEV

EZEDLLEV is set to zero under the following conditions:
- Initially
- When the PSB is terminated
- When another program is called

EZEDLLEV is reset each time a DL/I call is issued. It can be displayed to assist in error determination, or tested in the function error routine to determine the outcome of a DL/I call.

EZEDLLEV is read-only; it cannot be reset by the program.

EZEDLLEV is not set following CSPTDLI service calls or DL/I calls to IMS message queues or GSAM files.

Target environments for EZEDLLEV

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	Not supported.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.

Environment	Compatibility considerations
Solaris	Not supported.
CICS for Solaris	Not supported.
CICS for AIX	Not supported.
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Test Facility	None.

Example for EZEDLLEV

```
IF EZEDLLEV = 4;
END;
```

EZEDLPCB (DL/I)

EZEDLPCB is an array that represents PCBs in the DL/I PSB used by the program.

You specify EZEDLPCB with a literal subscript that identifies the specific PCB being used. The literal can range from 0 (the I/O PCB) to the highest PCB number for the PSB being used by the program.

The default subscript is 1.

Uses

You can use EZEDLPCB as any of the following:
- The source operand in a MOVE or assignment statement
- An argument on a CALL statement
- A parameter in a called parameter list

The characteristics of EZEDLPCB follow:

Data type
Character

Data length in bytes
Varies with the environment

Value saved across segments
No

You can use EZEDLPCB to do any of the following:

- Check the contents of the I/O PCBs and GSAM PCBs that are not accessible through the special function words that represent the contents of the database PCBs. A program can move the current contents of any PCB to a data item. The type of the source PCB and the length of the target area are used to determine how much of the PCB to move. If the source PCB is longer than the target area, the data is truncated. If the source PCB is shorter than the target area, the entire PCB is moved and the remaining target area is padded with blanks.
- Pass individual PCBs to a called program. The PSB structures of the main program and the called program can differ.

  You select which PCBs to pass to the called program by providing numeric literal subscripts on EZEDLPCB. The subscript identifies which PCB in the calling program's PSB that is to be passed to the called program.
- Specify the PCB parameters in the called parameter list of the called program. The subscript identifies the PCB definition in the called program's PSB that defines the structure of the PCB being passed. The PCB type and database definition for the PCB being passed must match the definition of the PCB in the called program.
- You cannot pass EZEDLPCB on a remote call.
- You cannot pass EZEDLPCB and EZEDLPSB on the same CALL statement.

Definition considerations for EZEDLPCB

You can move the contents of a PCB to the appropriate record for the PCB type and test the contents of the data items in the record definition. The sample PCB records are:

PCB TYPE	RECORD NAME	LOW-LEVEL ITEM NAME
I/O PCB	DLIIOPCB	IOPCB
Alternate	DLIALPCB	ALPCB
Database	DLIDBPCB	DBPCB
GSAM	DLIGSPCB	GSPCB

Target environments for EZEDLPCB

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	If EZEDLPCB is passed on a CALL statement and the PSB is not scheduled, the PSB is scheduled on the call. Moves from I/O, TP, and GSAM PCBs are not supported. Passing these PCBs as parameters is not supported.

Environment	Compatibility considerations
MVS/TSO	If EZEDLPCB is used in the called parameter list in a program called by a non-VisualAge Generator program, then PCB 0 must also be passed to the called program so that the called program can perform error recovery processing.
MVS batch	Same as MVS/TSO.
IMS/VS	If EZEDLPCB is used in the called parameter list in a program called by a non-VisualAge Generator program, then PCBs 0, 1, and 2 must also be passed to the called program so that the called program can perform error recovery processing. Either EZEDLPCB or EZEDLPSB must be passed as a parameter to a called program that issues DL/I calls. This includes the use of CSPTDLI or GSAM files.
IMS BMP	Same as MVS/TSO.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	Same as CICS for MVS/ESA.
CICS for OS/2	Not supported.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
CICS for AIX	Not supported.
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.

Environment	Compatibility considerations
Test Facility	The test facility allows EZEDLPCB to be passed on a call to another program loaded in the workspace/image. When EZEDLPSB is passed to another program loaded in the workspace/image, EZEDLPCB is passed as a null pointer (4 bytes). When EZEDLPCB[n] is moved to a data item, the PCB is converted from EBCDIC to ASCII according to the target data item definition. For any field you want to reference from the PCB, you must define a subfield in the target data item with the appropriate length, type, and offset. Refer to the sample programs for examples of records defined for IO, TP, DB, and GSAM PCBs. The Key Feedback area is converted by default according to the data item you have defined in your record. If your Key Feedback area contains multiple data types, you must substructure the key feedback item with items appropriate for each of your keys. I/O or TP PCB references using EZEDLPCB cannot be tested or emulated on the workstation.

Examples for EZEDLPCB

The following moves the I/O PCB:

```
MOVE EZEDLPCB[0] TO IOPCB;
```

The following example demonstrates passing individual PCBs on a CALL statement.

There are two main programs, APPLA and APPLB, and two called programs, APPLC and APPLD. The following PSBs are defined for the programs in the examples:

APPLA PSB A	APPLB PSB B	APPLC PSB C	APPLD PSB D
PCBB[1]-Parts	PCBC[1]-Order	PCBD[1]-Parts	
PCBB[2]	PCBC[2]-Parts	PCBD[2]-Journal	
PCBB[3]	PCBC[3]-Journal	PCBD[3]-Order	
PCBB[4]-Order	PCBC[4]		
PCBB[5]-Journal			

The following is the called parameter list for APPLC and APPLD:

EZEDLPCB[1]
EZEDLPCB[2]

- When APPLA calls APPLC to pass PCB number 3 and 5, the CALL statement is the following:

  ```
  CALL APPLC EZEDLPCB[3],EZEDLPCB[5];
  ```

  When the CALL statement is run, the following is true:
  PCBC[1] is associated with PCBA[3]
  PCBC[2] is associated with PCBA[5]
  PCBC[3] and PCBC[4] are not available for use.
- When APPLA calls APPLD to pass PCB number 5 and 2, the CALL statement is the following:

  ```
  CALL APPLD EZEDLPCB[5], EZEDLPCB[2];
  ```

  When the CALL statement is run, the following is true:
  PCBD[1] is associated with PCBA[5]
  PCBD[2] is associated with PCBA[2]
  PCBD[3] is not available for use.
- When APPLB calls APPLC to pass PCB number 4 and 1, the CALL statement is the following:

  ```
  CALL APPLC EZEDLPCB[4], EZEDLPCB[1];
  ```

  When the CALL statement is run, the following is true:
  PCBC[1] is associated with PCBB[4]
  PCBC[2] is associated with PCBB[1]
  PCBC[3] and PCBC[4] are not available for use.
- When APPLB calls APPLD to pass PCB number 1 and 5, the CALL statement is the following:

  ```
  CALL APPLD EZEDLPCB[1], EZEDLPCB[5];
  ```

  When the CALL statement is run, the following is true:
  PCBD[1] is associated with PCBB[1]
  PCBD[2] is associated with PCBB[5]
  PCBD[3] is not available for use.

EZEDLPRO (DL/I)

EZEDLPRO contains the DL/I options for the database accessed by the last DL/I I/O function. The options come from the database PCB used by the DL/I call for the function.

Uses

You can use EZEDLPRO as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

- An operand on a TEST statement

The characteristics of EZEDLPRO follow:

Data type
Character

Data length in bytes
4

Value saved across segments
No

Definition considerations for EZEDLPRO

EZEDLPRO is set to blanks under the following conditions:
- Initially
- When the PSB is terminated
- When another program is called

EZEDLPRO is reset to the current PCB values each time a DL/I call is issued. It can be displayed to assist in error determination, or tested in the function error routine to determine the outcome of a DL/I call.

EZEDLPRO is read-only; it cannot be reset by the program.

EZEDLPRO is not set following CSPTDLI service calls or DL/I calls to IMS message queues or GSAM files.

Target environments for EZEDLPRO

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	Not supported.
OS/400	Not supported.

Environment	Compatibility considerations
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
CICS for AIX	Not supported.
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Test Facility	None.

Example for EZEDLPRO

```
MOVE EZEDLPRO TO PRO2;
```

EZEDLPSB (DL/I)

EZEDLPSB contains the name of the DL/I PSB to be scheduled for DL/I database access. The default is the PSB name specified for the program.

When a program runs a DL/I function under CICS and a PSB is not currently scheduled, the program schedules the PSB named in EZEDLPSB.

Uses

You can use EZEDLPSB as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand in a MOVE or assignment statement
- An argument on a CALL statement
- A parameter in a called parameter list
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand on the TEST statement

The characteristics of EZEDLPSB follow:

Data type
Character

Data length in bytes
8

Value saved across segments
Yes

You can use EZEDLPSB to do any of the following:

- In CICS, change the name of the PSB to access a different set of databases for different users. If the program changes EZEDLPSB, the new PSB must describe databases with the same structure as the PSB for which the program was generated.

  A PSB is scheduled when the first DL/I I/O function of the program is executed. If EZEDLPSB changes, the new PSB is scheduled when the first DL/I I/O function is run after a COMMIT or ROLLBACK.
- Share a DL/I PSB with a called program by specifying the item as a parameter on a call to or from the program.

  When EZEDLPSB is passed as a parameter, the 8-byte name is followed by a 4-byte address field.

You cannot pass EZEDLPSB and EZEDLPCB on the same CALL statement.

Definition considerations for EZEDLPSB

To support portability of programs between IMS and non-IMS environments, EZEDLPSB can be included in the called parameter list of programs that do not include explicit DL/I processing functions (DL/I functions or CSPTDLI calls).

By using EZEDLPSB in the called parameter list, you can implement serial file processing using the PSB (message queues or GSAM) in IMS environments, and other facilities in non-IMS environments.

In non-DL/I programs, the EZEDLPSB parameter is ignored, so any argument can be passed as EZEDLPSB to the called program.

Target environments for EZEDLPSB

Environment	Compatibility considerations
VM CMS	Ignored.
VM batch	Ignored.

Environment	Compatibility considerations
CICS for MVS/ESA	When passing EZEDLPSB on a CALL, if the PSB is currently scheduled, the address field points to the CICS user interface block (UIB). If the PSB is not scheduled, the address field is 0. When the UIB address is passed to a program, the program accesses the PCB address list using the UIB. It does not attempt to reschedule the PSB.
MVS/TSO	EZEDLPSB is not used for PSB scheduling because PSBs are not scheduled in a non-CICS environment. A single PSB, specified at the invocation of the program, is used. Programs that run together using CALL, DXFR, and XFER must share the same PSB. The contents of EZEDLPSB are used as the checkpoint identifier field for the CHKP used to implement the EZECOMIT function in a DL/I batch environment. Either EZEDLPSB or EZEDLPCB must be passed as a parameter to a called program that issues DL/I calls. This includes the use of CSPTDLI or GSAM files. When EZEDLPSB is passed between a non-VisualAge Generator program and a program, the 4-byte address passed following the 8-byte PSB must point to a simulated CICS UIB. The simulated UIB is 6 bytes long: • The first 4 bytes must be the address of the PCB address list. • The last 2 bytes must contain binary zeros.
MVS batch	EZEDLPSB is not used for PSB scheduling because the PSB is specified in the JCL for the batch job step. Programs that run together in a single job step using CALL, DXFR, or XFER must share the same PSB. The contents of EZEDLPSB are used as the checkpoint identifier field for the CHKP used to implement the EZECOMIT function in a DL/I batch environment. Either EZEDLPSB or EZEDLPCB must be included in the called parameter list whenever a DL/I call occurs in this environment. This includes the use of CSPTDLI or GSAM files. When EZEDLPSB is passed between a non-VisualAge Generator program and a program, the 4-byte address passed following the 8-byte PSB must point to a simulated CICS UIB. The simulated UIB is 6 bytes long: • The first 4 bytes must be the address of the PCB address list. • The last 2 bytes must contain binary zeros.

Environment	Compatibility considerations
IMS/VS	EZEDLPSB is not used for PSB scheduling because IMS schedules the PSB automatically for IMS transactions. Programs that run together in a single IMS transaction using CALL or DXFR must share the same PSB. If a new value is moved into EZEDLPSB, the value is ignored. Either EZEDLPSB or EZEDLPCB must be passed as a parameter to a called program that issues DL/I calls. This includes the use of CSPTDLI or GSAM files. When EZEDLPSB is passed between a non-VisualAge Generator program and a program, the 4-byte address passed following the 8-byte PSB must point to a simulated CICS UIB. The simulated UIB is 6 bytes long: • The first 4 bytes must be the address of the PCB address list. • The last 2 bytes must contain binary zeros.
IMS BMP	EZEDLPSB is not used for PSB scheduling because the PSB was specified in the JCL for the batch job step. Programs that run together in a single job step using CALL, DXFR, or XFER must share the same PSB. For a batch-oriented BMP, the contents of EZEDLPSB are used as the checkpoint identifier on the CHKP call issued for EZECOMIT processing. For a transaction-oriented BMP, the contents of EZEDLPSB are ignored. Either EZEDLPSB or EZEDLPCB must be passed as a parameter to a called program that issues DL/I calls. This includes the use of CSPTDLI or GSAM files. When EZEDLPSB is passed between a non-VisualAge Generator program and a program, the 4-byte address passed following the 8-byte PSB must point to a simulated CICS UIB. The simulated UIB is 6 bytes long: • The first 4 bytes must be the address of the PCB address list. • The last 2 bytes must contain binary zeros.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	Same as MVS batch.
CICS for OS/2	If a program calls a remote server program that accesses DL/I databases, the calling program must pass EZEDLPSB as a parameter to the remote server program. The PSB is scheduled in the first remote server program, and the CICS UIB address is passed back in the EZEDLPSB parameter for use on subsequent calls to remote DL/I programs. Within a single unit of work, remote server programs must go to the same target system.

Environment	Compatibility considerations
OS/400	Ignored.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	Ignored.
AIX	Ignored.
HP-UX	Ignored.
Solaris	Ignored.
CICS for Solaris	If a program makes multiple calls within the same unit of work to a remote server program that accesses DL/I databases, the calling program must pass EZEDLPSB to the server program. The PSB is scheduled in the first server program, and the CICS UIB address is passed back in the EZEDLPSB parameter for use on subsequent calls to server DL/I programs. Within a single logical unit of work, all calls to DL/I remote server programs must go to the same target system.
CICS for AIX	If a program makes multiple calls within the same unit of work to a remote server program that accesses DL/I databases, the calling program must pass EZEDLPSB to the server program. The PSB is scheduled in the first server program, and the CICS UIB address is passed back in the EZEDLPSB parameter for use on subsequent calls to server DL/I programs. Within a single logical unit of work, all calls to DL/I remote server programs must go to the same target system.
Windows NT (C++)	Same as CICS for AIX.
Windows NT (Java)	Ignored.
CICS for Windows NT	Same as CICS for AIX.
Test Facility	EZEDLPSB can be passed on a call to another program loaded in the workspace/image. When EZEDLPSB is passed to another program loaded in the workspace/image, the 12 bytes passed include the PSB name currently in EZEDLPSB (8 bytes) and a NULL UIB pointer (4 bytes). Data passed back in the UIB pointer is preserved for subsequent calls, but is not used by the test facility.

Example for EZEDLPSB

```
CALL MYPROG EZEDLPSB;
```

EZEDLRST (DL/I)

EZEDLRST indicates whether the DL/I program has been restarted in an CICS for MVS/ESA or CICS for VSE/ESA environment following an abnormal end caused by a deadlock when queuing on database records.

Uses

You can use EZEDLRST as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- An occurrence operand in a MOVEA statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

The characteristics of EZEDLRST follow:

Data type
Numeric

Data length in bytes
1

Value saved across segments
No

Definition considerations for EZEDLRST

You cannot change the value of EZEDLRST in the program.

If EZEDLRST is equal to 1, the DL/I program has been restarted.

For special considerations about restarting your program, refer to the VisualAge Generator running manual for your operating environment.

Target environments for EZEDLRST

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	EZEDLRST is always set to 0.
MVS batch	Same as MVS/TSO.
IMS/VS	Same as MVS/TSO.

Environment	Compatibility considerations
IMS BMP	Same as MVS/TSO.
CICS for VSE/ESA	None.
VSE batch	Same as MVS/TSO.
CICS for OS/2	Same as MVS/TSO.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
CICS for AIX	Not supported.
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Test Facility	A CICS deadlock restart using EZEDLRST cannot be tested or emulated on the workstation.

Example for EZEDLRST

```
IF EZEDLRST = 1;
END;
```

EZEDLSEG (DL/I)

EZEDLSEG contains the name of the lowest-level segment found in the last DL/I I/O function. The name is from the database PCB used by the DL/I call for the function.

Uses

You can use EZEDLSEG as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement

- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand on the TEST statement

The characteristics of EZEDLSEG follow:

Data type
Character

Data length in bytes
8

Value saved across segments
No

Definition considerations for EZEDLSEG

EZEDLSEG is set to blanks under the following conditions:
- Initially
- When the PSB is terminated
- When another program is called

EZEDLSEG is reset to the current PCB values each time a DL/I call is issued. It can be displayed to assist in error determination, or tested in the function error routine to determine the outcome of a DL/I call.

EZEDLSEG is read-only; it cannot be reset by the program.

EZEDLSEG is not set following CSPTDLI service calls or DL/I calls to IMS message queues or GSAM files.

Target environments for EZEDLSEG

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.

Environment	Compatibility considerations
CICS for OS/2	Not supported.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
CICS for AIX	Not supported.
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Test Facility	None.

Example for EZEDLSEG

```
MOVE EZEDLSEG TO ITEM4;
```

EZEDLSSG (DL/I)

EZEDLSSG contains the number of segment types to which a program is sensitive for the database accessed during the last DL/I I/O function. The number is from the database PCB used by the DL/I call for the function.

Uses

You can use EZEDLSSG as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- An occurrence operand in a MOVEA statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

The characteristics of EZEDLSSG follow:

Data type
Binary

Data length in bytes
4

Value saved across segments
No

Definition considerations for EZEDLSSG

EZEDLSSG is set to zero under the following conditions:
- Initially
- When the PSB is terminated
- When another program is called

EZEDLSSG is reset to the current PCB values each time a DL/I call is issued. It can be displayed to assist in error determination, or tested in the function error routine to determine the outcome of a DL/I call.

EZEDLSSG is read-only; it cannot be reset by the program.

EZEDLSSG is not set following CSPTDLI service calls or DL/I calls to IMS message queues or GSAM files.

Target environments for EZEDLSSG

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	Not supported.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.

Environment	Compatibility considerations
Windows Java (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
CICS for AIX	Not supported.
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Test Facility	None.

Example for EZEDLSSG

```
MOVE EZEDLSSG TO ITEM5;
```

EZEDLSTC (DL/I)

EZEDLSTC contains the status code returned for the last DL/I I/O function. The code is from the database PCB used by the DL/I call for the function.

Uses

You can use EZEDLSTC as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand on the TEST statement

The characteristics of EZEDLSTC follow:

Data type
Character

Data length in bytes
2

Value saved across segments
No

Definition considerations for EZEDLSTC

EZEDLSTC is set to blanks when the following occurs:

- Initially
- When the PSB is terminated
- When another program is called

EZEDLSTC is reset to the current PCB values each time a DL/I call is issued. It can be displayed to assist in error determination, or tested in the function error routine to determine the outcome of a DL/I call.

EZEDLSTC is read-only; it cannot be reset by the program.

EZEDLSTC is not set following CSPTDLI service calls or DL/I calls to IMS message queues or GSAM files.

Target environments for EZEDLSTC

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	Not supported.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
Solaris	Not supported.

Environment	Compatibility considerations
CICS for Solaris	Not supported.
CICS for AIX	Not supported.
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Test Facility	None.

Example for EZEDLSTC

```
MOVE EZEDLSTC TO ITEM6;
```

EZEDLTRM (DL/I)

EZEDLTRM is a switch used to control whether data is automatically committed for every CONVERSE I/O option. When EZEDLTRM is set to 1, EZECOMIT is invoked during every CONVERSE. The default setting of EZEDLTRM is 0.

EZEDLTRM is equivalent to EZECNVCM. Setting one flag sets the other.

Uses

You can use EZEDLTRM as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand in a MOVE or assignment statement
- An occurrence operand in a MOVEA statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

The characteristics of EZEDLTRM follow:

Data type
Numeric

Data length in bytes
1

Value saved across segments
No

Definition considerations for EZEDLTRM

When EZECNVCM or EZEDLTRM is set to 1, EZECOMIT is automatically invoked during every CONVERSE function following terminal write, but before terminal read. This commits data changes to files or databases and logs terminal output at the same time. When EZECNVCM or EZEDLTRM is set to 0, a commit is done on the CONVERSE only if the program is running in segmented mode at the time of the CONVERSE.

Target environments EZEDLTRM

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	EZECNVCM and EZEDLTRM are ignored on a segmented CONVERSE. The CONVERSE marks the end of a segment; a commit is always done at the end of a segment.
MVS/TSO	None.
MVS batch	Ignored.
IMS/VS	The values of EZECNVCM and EZEDLTRM are ignored. In IMS, each CONVERSE is the end of a segment; a commit is always done at the end of a segment.
IMS BMP	Ignored.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	Ignored.
CICS for OS/2	Same as CICS for MVS/ESA.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
CICS for AIX	Not supported.
Windows NT (C++)	Not supported.

Environment	Compatibility considerations
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Test Facility	If the program is running under the test facility, data is committed only when a map appears.

EZEDTE

EZEDTE contains the system date in Gregorian format (YYMMDD). EZEDTE is automatically updated each time it is referenced by your program.

The retrieved date format is valid for use in variable fields defined with a Gregorian date edit mask with a two-digit year.

Uses

You can use EZEDTE as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement

The receiver can be a map field or data item.

The characteristics of EZEDTE follow:

Data type
Numeric

Data length in bytes
6

Value saved across segments
No

Target environments for EZEDTE

Supported in all environments without compatibility considerations.

Example for EZEDTE

```
MOVE EZEDTE TO MYDAY;
```

EZEDTEL

EZEDTEL retrieves the current date in Gregorian format (YYYYMMDD). EZEDTEL is automatically updated each time it is referenced by your program.

EZEDTEL

The Gregorian date is presented in a numeric format without separator characters.

The retrieved date format is valid for use in variable fields defined with a Gregorian date edit mask with a four-digit year.

Uses

You can use EZEDTEL as any of the following:

- A source operand on a MOVE, MOVEA, or assignment statement.

The receiver can be a map field or data item.

The characteristics of EZEDTEL follow:

Data type
Numeric

Data length in bytes
8

Value saved across segments
No

Target environments for EZEDTEL

Supported in all environments without compatibility considerations.

Example for EZEDTEL

```
MOVE EZEDTEL TO DATE_ITEM;
```

EZEDTELC

EZEDTELC retrieves the current date in the system default long Gregorian format. EZEDTELC is automatically updated each time it is referenced by your program.

For OS/2, if EZERGRGL_xxx does not exist, the default Gregorian format is derived from the OS/2 system settings. The system default format for the Gregorian date includes separator characters. The environment variable EZERGRGL_xxx where *xxx* determines the Gregorian format for dates.

The xxx specifies the language code. For example, the following are applicable language codes:

CHS Simplified Chinese

PTB Brazilian Portuguese

ENU English

JPN Japanese

KOR Korean

For VisualAge Generator Server for MVS, VSE, and VM environments, if EZERGRGL_xxx does not exist, the default Gregorian format is specified during installation.

The retrieved date format is valid for use in variable fields defined with a Gregorian date edit mask with a four-digit year.

Uses

You can use EZEDTELC as any of the following:
- A source operand on a MOVE, MOVEA, or assignment statement.

The receiver can be a map field or data item.

The characteristics of EZEDTELC follow:

Data type
Character

Data length in bytes
10

Value saved across segments
No

Target environments for EZEDTELC

Supported in all environments without compatibility considerations.

Example for EZEDTELC

```
MOVE EZEDTELC TO DATE_ITEM;
```

EZEFEC

EZEFEC controls whether a program continues to run after a hard I/O error occurs on a function I/O operation for a file, database, or message queue record. A hard I/O error is any error except record not found, end of file, or duplicate record. For a description of the hard I/O errors, see "I/O error value" on page 368.

If EZEFEC is set to 1 and a function error routine has been specified, the function error routine runs when a hard I/O occurs. The program is responsible for reporting the error to the program user. The program can test the HRD record status to determine if a hard error occurred. EZERT8 contains a file dependent error code describing the error. EZERT2 also contains the MQSeries completion code if the record is a message queue record.

EZEFEC

If EZEFEC is set to 0, the program ends with an error message when a hard error occurs on a record I/O.

The initial setting is 0.

If you are using the /ANSISQL generation option, an SQLCODE is treated as though it were DB2/VSE or DB2 codes. To treat them differently, set EZEFEC to 1 and include an error routine for the I/O option.

Uses

You can use EZEFEC as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand in a MOVE or assignment statement
- An occurrence operand in a MOVEA statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

The characteristics of EZEFEC follow:

Data type
Numeric

Data length in bytes
1

Value saved across segments
Yes

Target environments for EZEFEC

Supported in all environments without compatibility considerations.

Example for EZEFEC

```
MOVE 1 to EZEFEC;
```

EZEFLO

EZEFLO causes control to transfer to the flow statements specified for the current main function.

If EZEFLO is specified as the error routine of a function and an I/O error occurs, processing continues with the first flow statement for the current main function.

Uses

You can use EZEFLO as any of the following:
- A function error routine name
- The true or false operand in a TEST or FIND statement in a function

- The function name in a function invocation statement

Target environments for EZEFLO

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
Solaris	None.
CICS for Solaris	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	None.
CICS for Windows NT	None.

Environment	Compatibility considerations
Test Facility	If EZEFLO or EZECLOS is encountered and Exit breakpoints are set on any of the functions that are currently listed in the Execution Stack Monitor, the following status message is displayed in the status area of the Test Monitor: `Exit Breakpoints exist` The break in execution will occur on the EZEFLO or EZECLOS statement.

Example for EZEFLO

```
IF MY_RECORD IS NRF;
   EZEFLO;
END;
```

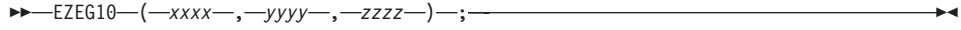
EZEG10

EZEG10 generates a modulus 10 check-digit.

Uses

You can use EZEG10 as the function name in a function invocation statement.

The calling sequence for EZEG10 is:



xxxx
A character data item in working storage that contains the number for which you want to generate a check digit, including a position for the check digit.

yyy
A binary ydata item of less than 5 digits that contains the number of characters to be used in item xxxx, including the check digit.

zzzz
A binary data item of less than 5 digits that returns a 0 if the modulus 10 check digit was successfully generated and a 1 if the check digit was not generated.

Target environments for EZEG10

Supported in all environments without compatibility considerations.

Example for EZEG10

In the following example, myinput is defined as character data containing the value 1734280 (the rightmost 0 is the position for the generated digit and can

be any value on input), mylength is a binary data item containing the value 7, and myresult is a binary data item whose value will be set by the EZEG10 routine.

```
EZEG10(myinput,mylength,myresult);
```

EZEG10 derives the modulus-10 check digit, using the following algorithm:

1. Multiply the units position of the base number of source data by 2, and multiply every alternate position, moving right to left, by 2:

   ```
   1    7    3    4    2    8    0 (generated digit)
      x 2       x 2       x 2
      ___       ___       ___
       14         8        16
   ```

2. Add the digits of the products to the digits of the base number that were not multiplied by 2:

   ```
   1 + 1 + 4 + 3 + 8 + 2 + 1 + 6 = 26
   ```

3. Subtract the sum from the next-highest number ending in 0 to get the self-checking digit. (If the difference is 10, 0 is used.)

   ```
   30 - 26 = 4
   ```

   The resulting digit is the self-checking digit. The variable myresult is set to 0 if the self-checking digit is generated or 1 if not.

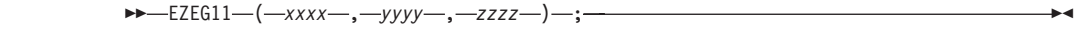
EZEG11

EZEG11 generates a modulus 11 check-digit.

Uses

You can use EZEG11 as the function name in a function invocation statement.

The calling sequence for EZEG11 is:



xxxx
A character data item in working storage that contains the number for which you want to generate a check digit, including a position for the check digit.

yyyy
A binary data item of less than 5 digits that contains the number of characters to be used in item xxxx, including the check digit.

zzzz
A binary data item of less than 5 digits that returns a 0 if the modulus 11 check digit was successfully generated and a 1 if the check digit was not generated.

Target environments for EZEG11

Supported in all environments without compatibility considerations.

Example for EZEG11

In the following example, `myinput` is defined as character data containing the value 56621865 (the rightmost 5 is the entered self-checking digit, not part of the base number), `mylength` is a binary data item containing the value 8, and `myresult` is a binary data item whose value will be set by the EZEG11 routine.

```
EZEG11(myinput,mylength,myresult);
```

EZEG11 derives the modulus-11 check digit using the following algorithm:

1. Multiply the units (rightmost) digit of the base number by 2, the tens position by 3, the hundreds position by 4, and so on, until 7 is used as a multiplier. If there are more digits to multiply, begin the sequence again using 2 as a multiplier:

```
  5    6    6    2    1    8    6    5 (Self-checking digit)
x 2  x 7  x 6  x 5  x 4  x 3  x 2
 10   42   36   10    4   24   12
```

2. Add the products of step 1:

```
10 + 42 + 36 + 10 + 4 + 24 + 12 = 138
```

3. Divide the sum of the products by 11:

```
      12
   _______
11 |138
     11
    ____
     28
     22
    ____
      6
```

Subtract the remainder from 11 to get the self-checking digit. (If the remainder is 0 or 1, 0 is used.)

```
11 - 6 = 5
```

The resulting digit is the self-checking digit. The variable `myresult` is set to 0 if the self-checking digit is generated or 1 if not.

EZELOC

EZELOC contains the system identifier for the location of a remote program or file.

You can use the linkage table to specify that a CICS program invoked by CALL or CREATX, or a CICS VSAM file or transient data queue is located on a remote system. The linkage table specifies whether the location of a remote file or program is specified in the CICS tables or whether it is to be obtained at run time from the EZELOC special function word. If you specify EZELOC in the linkage table, the program can dynamically modify EZELOC to allow selection of different locations for the remote program or file.

Uses

You can use EZELOC as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand of a MOVE or assignment statement
- Data item 1 and 2 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand on the TEST statement

The characteristics of EZELOC follow:

Data type
Character

Data length in bytes
8

Value saved across segments
Yes

Definition considerations for EZELOC

The program or file must be defined before run time in the CICS tables on all systems on which the program or file can reside or be accessed.

EZELOC does not support dynamic definition of files or programs, but does support dynamic selection from a predefined set of possible locations for the file or program.

EZELOC is initialized to blanks and must be set before doing any CALL, CREATX, or file I/O function that requires the location. If different locations are used for different functions, code the program to move the correct value to EZELOC immediately prior to the execution of the function that uses it.

Prior to its use, the value in EZELOC is folded to uppercase. However, the value in EZELOC does not change.

The special function word EZELOC will test true when compared against the lowercase version if that is how it was initialized.

Target environments for EZELOC

Environment	Compatibility considerations
VM CMS	Not supported. EZELOC has no effect because access to remote programs and files is not supported.
VM batch	Same as VM CMS.
CICS for MVS/ESA	None.
MVS/TSO	Same as VM CMS.
MVS batch	Same as VM CMS.
IMS/VS	Same as VM CMS.
IMS BMP	Same as VM CMS.
CICS for VSE/ESA	None.
VSE batch	Same as VM CMS.
CICS for OS/2	None.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	None.
Windows Java (GUI)	None.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
Solaris	Not supported.
CICS for Solaris	None.
CICS for AIX	None.
Windows NT (C++)	Not supported.
Windows NT (Java)	None.
CICS for Windows NT	None.
Test Facility	None.

Example for EZELOC

```
MOVE 'SYSTEMA' TO EZELOC;
```

EZELTERM

EZELTERM contains the terminal identifier, if one exists in the environment where the program is running. The terminal identifier is padded with blanks to 8 characters. EZELTERM is equivalent to EZEUSR in the CICS environments.

For Web Transaction programs, EZELTERM contains the user session id code. Use the EZELTERM value as a key value to save program state in a file or database across a CONVERSE or XFER to a program with UI record. Session boundaries are determined by the type of user event. Clicking on a program link starts a new session with a new session code; pressing a button activates a program within the same session.

Uses

You can use EZELTERM as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand on the TEST statement

The characteristics of EZELTERM follow:

Data type
Character

Data length in bytes
8 (padded with blanks)

Value saved across segments
Yes

Target environments for EZELTERM

Environment	Compatibility considerations
VM CMS	EZELTERM is blank.
VM batch	Same as VM CMS.
CICS for MVS/ESA	EZELTERM contains the CICS terminal identifier. EZELTERM is equivalent to EZEUSR.
MVS/TSO	Same as VM CMS.
MVS batch	Same as VM CMS.

Environment	Compatibility considerations
IMS/VS	EZELTERM contains the logical terminal identifier from the first 8 bytes of the I/O PCB. EZELTERM is updated whenever there is a successful get unique call to the I/O PCB. This is caused by a SCAN for a serial file associated with the I/O PCB, a CONVERSE I/O option, or a first map. EZELTERM is set to blanks when a main batch program that scans the message queue gets an EOF (GC status code for a get unique call).
IMS BMP	EZELTERM is initialized to blanks. If the program runs as an IMS transaction-oriented BMP, EZELTERM is reset to the logical terminal identifier from the first 8 bytes of the I/O PCB on each SCAN that results in a successful get unique call for a serial file associated with the I/O PCB.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	Same as VM CMS.
CICS for OS/2	Same as CICS for MVS/ESA.
OS/400	EZELTERM is initialized to blanks. If the program is run in an OS/400 interactive job, EZELTERM is reset to the terminal device name received from a query of the active job's attributes. On OS/400, EZELTERM is 10 characters in length, padded with blanks.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	EZELTERM is equivalent to EZEUSR.
AIX	In the AIX environment, the first directory specification is ignored, since terminal names can be arbitrarily long. From there, the first 8 characters are returned as the terminal identifier. For example, if the terminal name returned from the AIX "tty" command is: `/dev/pts/1` The value of EZELTERM is "pts/1".
HP-UX	In the HP-UX environment, the first directory specification is ignored, since terminal names can be arbitrarily long. From there, the first 8 characters are returned as the terminal identifier. For example, if the terminal name returned from the HP-UX "tty" command is: `/dev/pts/1` The value of EZELTERM is "pts/1".

Environment	Compatibility considerations
Solaris	In the AIX Solaris environment, the first directory specification is ignored, since terminal names can be arbitrarily long. From there, the first 8 characters are returned as the terminal identifier. For example, if the terminal name returned from the AIX Solaris"tty" command is: `/dev/pts/1` The value of EZELTERM is "pts/1".
CICS for Solaris	Same as CICS for MVS/ESA.
CICS for AIX	Same as CICS for MVS/ESA.
Windows NT (C++)	Same as OS/2 (C++).
Windows NT (Java)	Same as OS/2 (C++).
CICS for Windows NT	Same as CICS for MVS/ESA.
Test Facility	None.

Example for EZELTERM

```
MOVE EZELTERM TO ITEM10;
```

EZEMNO

EZEMNO sets the number for the message that appears on the next CONVERSE, DISPLAY, XFER with a map, or redisplay of a map with an edit error.

The text of the message is taken from the program message table. The message text appears in the variable field EZEMSG on the map.

EZEMNO is initialized and reset to 0 after every CONVERSE, DISPLAY, XFER with a map, or redisplay of a map for an edit error. On the next CONVERSE, DISPLAY, XFER with a map, or redisplay of a map for an edit error, if EZEMNO contains a value other than 0 or 9999, the message specified by that number is retrieved and the map is then redisplayed. The range for EZEMNO is from -9999 to 9999.

Uses

You can use EZEMNO as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand in a MOVE or assignment statement
- An occurrence operand in a MOVEA statement

- Data item 1 or 2 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

The characteristics of EZEMNO follow:

Data type
Binary

Data length in bytes
2

Value saved across segments
No

Definition considerations for EZEMNO

A function used as an edit routine indicates that an error has been detected by moving a nonzero value to EZEMNO. This automatically displays the map again, with the field in error highlighted and the text of the message displayed.

If a message table is not available, an edit routine can force the map to be conversed again by moving message text to EZEMSG and setting EZEMNO to 9999.

Target environments for EZEMNO

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
Windows & OS/2 Smalltalk (GUI)	Not supported.

Environment	Compatibility considerations
Windows Java (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
Solaris	None.
CICS for Solaris	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	Not supported.
CICS for Windows NT	None.
Test Facility	None.

Examples for EZEMNO

The following example sets EZEMNO to 9999. You could use this example in an edit routine to cause the map to be conversed again.

```
MOVE 'There has been an error' TO EZEMSG;
MOVE 9999 TO EZEMNO;
```

The following example causes the text for message 1234 to be retrieved from the program message table before the next map is conversed.

```
MOVE 1234 to EZEMNO;
```

EZEMSG

EZEMSG is used for displaying message text on the next CONVERSE, DISPLAY, XFER with a map, or redisplay of a map with an edit error.

To display a message on a map, you need to define EZEMSG as a variable field on the map. If you do not define an EZEMSG field, the screen is cleared before the error message is displayed to the program user. When the program user presses the Enter key, the program map appears again.

Uses

You can use EZEMSG as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand in a MOVE or assignment statement
- Data item 1 or 2 in a RETR statement

- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand on the TEST statement
- A map item in a SET statement

The characteristics of EZEMSG follow:

Data type
Character or Mixed

Data length in bytes
11 to 78

Value saved across segments
No

Definition considerations for EZEMSG

EZEMSG for a map is set to spaces after the map is conversed or displayed, or when a *SET map CLEAR* is performed. EZEMSG is used as a message area for VisualAge Generator Server for MVS, VSE, and VM or VisualAge Generator Server editing messages. However, literals can be moved to EZEMSG if EZEMNO contains 0 or 9999 and *SET map CLEAR* is not specified after the MOVE.

Target environments for EZEMSG

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
Windows & OS/2 Smalltalk (GUI)	Not supported.

Environment	Compatibility considerations
Windows Java (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
Solaris	None.
CICS for Solaris	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	Not supported.
CICS for Windows NT	None.
Test Facility	None.

Example for EZEMSG

```
MOVE 'Invalid data' to EZEMSG;
```

EZEOVER

EZEOVER controls error processing after an arithmetic overflow.

Two types of overflow conditions are detected:

Maximum value
Occurs when the result of an arithmetic operation is greater than 18 digits.

User variable
Occurs when the result of an arithmetic operation or a move to a numeric data item causes a significant value (not decimal positions) to be lost due to the length of the data item.

Depending on the value of EZEOVER, the overflow condition is handled differently. You can set EZEOVER to one of the following values. The default setting is 0.

0 On maximum value overflow, the program ends abnormally with an error message.

On user variable overflow, the program continues and special function word EZEOVERS is set to 1.

1 Ends the program when there is either a maximum value or user variable overflow. An error message is issued indicating the statement that caused the overflow condition.

2 Continues to run the program when a maximum value or user variable overflow occurs and special function word EZEOVERS is set to 1.

Uses

You can use EZEOVER as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand in a MOVE or assignment statement
- An occurrence operand in a MOVEA statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

The characteristics of EZEOVER follow:

Data type
Numeric

Data length in bytes
1

Value saved across segments
Yes

Target environments for EZEOVER

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.

Environment	Compatibility considerations
Windows & OS/2 Smalltalk (GUI)	None.
Windows Java (GUI)	None.
OS/2 (C++)	The generated C++ code can contain more precision than the COBOL programs. In calculating significant digits, zeros to the left of the decimal point are not considered significant digits. For example, in the following expression: `x = (999999999999999999 + 1)/5;` The intermediate result of the parenthetical expression is 1 x 10**18. This number is considered to have 1 significant digit, not 19. Therefore, the calculation can continue without an overflow error. **Note:** The NUMOVFL generation option is not supported in the C++ program generator. Overflow checking always occurs.
AIX	Same as OS/2 (C++).
HP-UX	Same as OS/2 (C++).
Solaris	Same as OS/2 (C++).
CICS for Solaris	Same as OS/2 (C++).
CICS for AIX	Same as OS/2 (C++).
Windows NT (C++)	Same as OS/2 (C++).
Windows NT (Java)	None.
CICS for Windows NT	Same as OS/2 (C++).
Test Facility	A warning is issued on maximum value overflow.

Example for EZEOVER

```
MOVE 2 TO EZEOVER;
```

EZEOVERS

EZEOVERS provides the means to test for arithmetic overflow. EZEOVERS is set to 1 to indicate that arithmetic overflow has occurred.

After an overflow condition is detected, EZEOVERS is not reset automatically. You need to code the program to reset EZEOVERS to 0 before doing any calculations or moves that you want checked for arithmetic overflow.

EZEOVERS

Uses

You can use EZEOVERS as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand in a MOVE or assignment statement
- An occurrence operand in a MOVEA statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

The characteristics of EZEOVERS follow:

Data type
Numeric

Data length in bytes
1

Value saved across segments
Yes

Target environments for EZEOVERS

Supported in all environments without compatibility considerations.

Example for EZEOVERS

```
MOVE 0 TO EZEOVERS;
MOVE 2 TO EZEOVER;
A = B * C;
IF EZEOVERS EQ 1;
   MOVE 1234 to EZEMNO;
   CONVERSE-MYMAP();
END;
```

EZEPURGE

EZEPURGE deletes a CICS temporary storage queue.

Uses

You can use EZEPURGE as the function name in a function invocation statement.

The calling sequence is:

Attribute	Description
queuename	Is a 1- to 8-character data item or 1- to 8-byte character literal that contains the name of a single temporary storage queue to be deleted.

You must provide a queue name on the call to let the program know which queue is to be deleted from temporary storage. The program enqueues (ENQ command with the NOSUSPEND option) on the resource name EZETEMP-queuename on an EZEPURGE invocation. The program dequeues (DEQ command) after the temporary storage queue is deleted.

If an error occurs, the first byte of the EIBFN is placed in the first 2 characters of EZERT8, and bytes 0 to 2 of the EIBRCODE are placed in the last 6 characters of EZERT8.

Target environments for EZEPURGE

Environment	Compatibility considerations
VM CMS	Ignored.
VM batch	Ignored.
CICS for MVS/ESA	None.
MVS/TSO	Ignored.
MVS batch	Ignored.
IMS/VS	Ignored.
IMS BMP	Ignored.
CICS for VSE/ESA	None.
VSE batch	Ignored.
CICS for OS/2	None.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
CICS for AIX	None.
Solaris	Not supported.
CICS for Solaris	None.
Windows NT (C++)	Not supported.

Environment	Compatibility considerations
Windows NT (Java)	Not supported.
CICS for Windows NT	None.
Test Facility	Not supported.

Examples for EZEPURGE

The following deletes the CICS temporary storage queue associated with the current value of EZEDEST for record ABC:

```
MOVE ABC.EZEDEST to MYQUEUE;
 EZEPURGE (MYQUEUE);
```

Where MYQUEUE is a data item name.

The following deletes the CICS temporary storage queue associated with destination XYZ:

```
 EZEPURGE('XYZ');
```

EZERCODE

EZERCODE is an external return code to be checked by the JCL, command processor, or calling high-level language program when the program ends. Passing return codes from one program to another is not supported.

EZERCODE is initially set to 0. If the program ends because of an unexpected error, EZERCODE is set to a value greater than 512. Programs should not set EZERCODE to a value greater than 512 and should not set negative return codes.

EZERCODE is implemented using the COBOL RETURN-CODE special register. The contents of EZERCODE are used to set RETURN-CODE when a program ends.

If the link type for a called program is static or dynamic, the return code is returned in register 15 on System/370 processors. The return code is not passed back for a called remote, called CICS :link, or main program.

You can use EZERCODE as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand in a MOVE or assignment statement
- An occurrence operand in a MOVEA statement
- Data item 1 or 2 in a RETR statement
- A data item in an IF or WHILE statement

- A data item in a FIND statement

The characteristics of EZERCODE follow:

Data type
Binary

Data length in bytes
4

Value saved across segments
Yes

Target environments for EZERCODE

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	If the link type for a called program is static or dynamic, the return code is passed back and placed in Register 15. The return code is not passed back for a called remote, called CICS :link, or main program.
MVS/TSO	None.
MVS batch	None.
IMS/VS	The return code is not passed back from main programs.
IMS BMP	None.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	None.
CICS for OS/2	Same as CICS for MVS/ESA.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	If the program ends abnormally, a return code other than EZERCODE is returned. In AIX, OS/2 (C++), and Windows NT this return code is 65280.
AIX	Same as OS/2 (C++).
HP-UX	Same as OS/2 (C++).
Solaris	Same as OS/2 (C++).

Environment	Compatibility considerations
CICS for Solaris	The value in EZERCODE is not passed back to the system or calling program.
CICS for AIX	The value in EZERCODE is not passed back to the system or calling program.
Windows NT (C++)	Same as OS/2 (C++).
Windows NT (Java)	Ignored. The return code of a program is dependent on the JVM.
CICS for Windows NT	Same as CICS for AIX.
Test Facility	None.

Example for EZERCODE

```
EZERCODE = 6;
```

EZEREPLY

EZEREPLY specifies whether any exception code raised by the VAGen supplied function should be returned in EZERT8. EZEREPLY is only effective for the VAGen supplied functions.

If EZEREPLY is set to 1 when a VAGen supplied function fails, the return code indicates why the function invocation failed and is available in the EZERT8 special function word as the 8-character displayable form of the return code. Execution continues with the statement immediately following the function invocation.

If EZEREPLY is set to 0 when a VAGen supplied function fails, the invoking program ends with an error message that explains the reason for the termination and displays the return code.

The initial value of EZEREPLY is 0.

The setting of EZEREPLY has no effect on:
- EZESBLKT
- EZESCCWS
- EZESCNCT
- EZESTLEN
- EZECONCT
- EZEC10
- EZEC11
- EZECONV

- EZEG10
- EZEG11
- EZEPURGE
- EZEWAIT

Uses

You can use EZEREPLY as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand in a MOVE or assignment statement
- An occurrence operand in a MOVEA statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

The characteristics of EZEREPLY follow:

Data type
Numeric

Data length in bytes
1

Value saved across segments
No

Target environments for EZEREPLY

Supported in all environments without compatibility considerations.

Example for EZEREPLY

```
MOVE 1 to EZEREPLY;
```

EZEROLLB

EZEROLLB calls system services to back out recoverable file, database, and message queue updates since the last commit point. VAGen issues an environment rollback request when EZEROLLB is processed. A rollback occurs automatically if the program ends with an unexpected error.

Uses

You can use EZEROLLB as the function name in a function invocation statement.

You can use EZEROLLB in place of the CALL RESET service routine.

A roll back occurs when:
- A VisualAge Generator program calls the EZEROLLB or RESET service.
- A program ends because of an error condition.

Definition considerations for EZEROLLB

When you use EZEROLLB with message queue records, note the following:

- Message queue updates are recoverable only if the **Include message in transaction** option is selected in message queue record definition.
- Both message SCANs and ADDs are affected by commit and rollback for recoverable messages. If a rollback is issued following a SCAN for a recoverable message, the message is placed back on the input queue so that the input message is not lost when the transaction fails to complete successfully.

Target environments for EZEROLLB

Environment	Compatibility considerations
VM CMS	EZEROLLB results in an SQL ROLLBACK WORK if SQL requests have been issued by the program.
VM batch	Same as VM CMS.
CICS for MVS/ESA	EZEROLLB results in a CICS SYNCPOINT ROLLBACK. This rolls back any updates from the last commit point to DL/I databases, relational databases, and files defined as recoverable resources. EZEROLLB services are called automatically for the program if the program calls the RESET service. Remote called batch program can invoke EZEROLLB. Invoking EZEROLLB from remote programs result in an CICS SYNCPOINT ROLLBACK if ECI_NO_EXTENDED is specified in the middleware routing table. A runtime error message is issued indicating that rollback failed with INVREQ if ECI_EXTEND is specified in the middleware routing table. Spool files are rolled back.
MVS/TSO	EZEROLLB results in an SQL ROLLBACK WORK if the program has issued SQL requests. If DL/I requests have been issued, a DL/I ROLB (rollback) call is executed. The IMS batch parameter BKO=Y must be specified in the startup CLIST for the ROLB call to be honored. If BKO=N is specified, DL/I returns status code AL for the ROLB call. Execution treats the AL as a soft error code, and no error message is issued. VisualAge Generator programs that do not use DL/I issue a rollback only if the program has made changes to an SQL table. A rollback does not occur for changes to an SQL table made by a non-VisualAge Generator program.

Environment	Compatibility considerations
MVS batch	If the program runs under the TSO terminal monitor program for SQL access, invoking EZEROLLB results in an SQL ROLLBACK WORK. If the program runs as a DL/I batch job and DL/I or SQL requests have been issued, a DL/I ROLB call is issued. The IMS batch parameter BKO=Y must be specified when the batch job is started in order for the ROLB call to be honored. If BKO=N is specified, DL/I returns status code AL for the ROLB call. Execution treats the AL as a soft error, and no error message is issued. BKO is specified as a parameter in the job step that calls the IMS control program DFSRRC00. Serial or print files associated with GSAM files and CALL AUDIT result in DL/I requests and cause the DL/I ROLB call to be issued. VisualAge Generator programs that do not use DL/I issue a rollback only if the program has made changes to an SQL table. A rollback does not occur for changes to an SQL table made by a non-VisualAge Generator program.
IMS/VS	EZEROLLB results in a DL/I ROLB call.
IMS BMP	Same as IMS/VS.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	EZEROLLB results in an SQL ROLLBACK WORK if SQL requests have been issued by the program. There is no automatic rollback function for DL/I databases under VSE batch. If you need to roll back changes to a DL/I database, you must code your program to end with an error message indicating the end user should run the DL/I DOS/VS backout utility. Spool files are rolled back.

Environment	Compatibility considerations
CICS for OS/2	EZEROLLB results first in an SQL ROLLBACK WORK, if the program issued SQL requests on the workstation, and then a CICS SYNCPOINT ROLLBACK. CICS coordinates the host and workstation roll back functions. The CICS SYNCPOINT ROLLBACK backs out changes made to workstation files that are defined as recoverable resources to CICS OS/2. The CICS SYNCPOINT ROLLBACK also backs out changes to databases and recoverable files on a CICS host when the changes are made by a remote called batch program that is called by this program. The SQL ROLLBACK WORK backs out changes made to relational databases. Remote called batch program can invoke EZEROLLB. Invoking EZEROLLB from remote programs result in an CICS SYNCPOINT ROLLBACK if ECI_NO_EXTENDED is specified in the middleware routing table. A runtime error message is issued indicating that rollback failed with INVREQ if ECI_EXTEND is specified in the middleware routing table. Files generated with file type OS2COBOL are not recoverable resources and are not affected by EZEROLLB.
OS/400	None.
Windows & OS/2 Smalltalk (GUI)	EZEROLLB uses the client/server communication support.
Windows Java (GUI)	Same as Windows & OS/2 Smalltalk (GUI)..
OS/2 (C++)	EZEROLLB rolls back changes to relational databases. Files are not affected by EZEROLLB.
AIX	Same as OS/2 (C++).
HP-UX	Same as OS/2 (C++).
Solaris	Same as OS/2 (C++).
CICS for Solaris	EZEROLLB rolls back changes to relational databases and files defined as recoverable resources.
CICS for AIX	EZEROLLB rolls back changes to relational databases and files defined as recoverable resources.
Windows NT (C++)	Same as OS/2 (C++).
Windows NT (Java)	Same as OS/2 (C++).

Environment	Compatibility considerations
CICS for Windows NT	Same as CICS for AIX.
Test Facility	The VSE batch DL/I execution environment does not support EZEROLLB. The test facility issues EZEROLLB so that manual clean-up of the database is not necessary. If you are using the local DL/I emulation option, the Micro Focus products ignore a rollback request. Database positioning is lost when the rollback is issued, but the data itself is not rolled back.

Example for EZEROLLB

```
EZEREPLY=1;
EZEROLLB();
```

EZERTN

EZERTN causes an immediate return from a function.

Specifying EZERTN enables you to return a value from a function.

Specifying EZERTN in a main function causes an immediate transfer to the first flow statement specified for that function. EZERTN in a main function is equivalent to EZEFLO.

Specifying EZERTN as an error routine in a function causes the program to continue with the statement immediately following the I/O option if an I/O error occurs.

Uses

You can use EZERTN as any of the following:
- The name of a function error routine
- The true or false operand in a TEST or FIND statement in a function
- The function name in a function invocation statement

Target environments for EZERTN

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.

Environment	Compatibility considerations
IMS/VS	None.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
Windows & OS/2 Smalltalk (GUI)	None.
Windows Java (GUI)	None.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
Solaris	None.
CICS for Solaris	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	None.
CICS for Windows NT	None.
Test Facility	If EZERTN is encountered and an `Exit breakpoint` is set on the logic part that is being executed, the following status message is displayed in the status area of the Test Monitor: `Break on part - function_name` The break in execution will occur on the EZERTN statement.

Example for EZERTN

```
TEST ITEM3 BLANKS EZERTN;
EZERTN (ABC);
```

EZERT2

EZERT2 contains the completion code from an MQSeries API call following an ADD or SCAN I/O operation for a message queue record.

Valid values are:
00 OK
01 WARNING
02 FAILED

Uses

You can use EZERT2 as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand in a MOVE or assignment statement
- Data item 1 or 2 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand on the TEST statement

The characteristics of EZERT2 follow:

Data type
Character

Data length in bytes
2

Value saved across segments
Yes

Target environments for EZERT2

Supported in all environments without compatibility considerations.

EZERT8

EZERT8 contains the status code after:
- An I/O function to an indexed, message queue, relative or serial file
- CALL statements with the REPLY option
- Function invocation statements when EZEREPLY equals 1

Uses

You can use EZERT8 as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand in a MOVE or assignment statement
- Data item 1 or 2 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand on the TEST statement

The characteristics of EZERT8 follow:

Data type
Character

Data length in bytes
8

Value saved across segments
Yes

Definition considerations for EZERT8

The following sections address server call errors and the implications of using EZERT8 with message queues.

Server call error considerations

If an error occurs on a call to a server program, the program behaves differently depending on the type of error and whether the REPLY option was coded on the CALL statement.

Table 22. Error handling for remote calls

Type of Error	REPLY Option Specified	EZERT8 Setting	Log or Trace Message on Client	Log or Trace Message on Server	Program Ends With Message to User
No error, successful completion	No	Unchanged	No	No	No
No error, successful completion	Yes	0	No	No	No
Communication failure or terminating error in server	No	Unchanged	Yes	No	Yes
Communication failure or terminating error in server	Yes	CSO error message number	Yes	No	No

Message queue record I/O considerations

EZERT8 contains the MQ API call reason code following on I/O operation for a message queue record.

Generation Considerations for EZERT8

You can use the /SYSCODES generation option to control the codes that are returned for file I/O function errors.

If you specify /SYSCODES, EZERT8 contains system file I/O status codes. The codes are dependent on run-time environment and file type.

Target environments for EZERT8

Environment	Compatibility considerations
VM CMS	If EZERT8 is in the form `RSnnnnnn`, look under `nnnnnn` in the return codes section of *VisualAge Generator Server Guide for MVS, VSE, and VM* **SEQ** EZERT8 contains the COBOL status key value or values in the first 2 characters. The remaining 6 characters are zeros. **SEQRS** The contents of EZERT8 depend on the operation that failed: • If a dynamic allocation fails, EZERT8 contains the value '00000218'. • If an OPEN fails, EZERT8 contains return code 8 ('00000008'). • If a READ end-of-file condition occurs, EZERT8 contains return code 4 ('00000004'). • If a READ, WRITE, or CLOSE fails, EZERT8 contains return code 12 ('00000012'). **VSAM** EZERT8 contains the COBOL status key value or values in the first 2 characters followed by 2 characters for the COBOL VSAM return code (VSAM feedback code), 1 character for the COBOL VSAM function code (VSAM component code), and 3 characters for the COBOL VSAM feedback code (VSAM reason code). **VSAMRS** The operation that fails determines the contents of EZERT8: • If a dynamic allocation fails, EZERT8 contains the value '00000218'. • If an OPEN or CLOSE fails, the first 2 bytes of EZERT8 contain the error code from the VSAM program control block (ACB) in hexadecimal. The remaining 6 characters are zeros. • If an operation other than OPEN or CLOSE fails, the first 2 characters are zeros followed by 2 characters for the COBOL VSAM return code (VSAM feedback code), 1 character for the COBOL VSAM function code (VSAM component code), and 3 characters for the COBOL VSAM feedback code (VSAM reason code).

Environment	Compatibility considerations
VM batch	If EZERT8 is in the form RSnnnnnn, look under nnnnnn in the return codes section of *VisualAge Generator Server Guide for MVS, VSE, and VM* **SEQ** Same as VM CMS. **SEQRS** Same as VM CMS. **VSAM** Same as VM CMS. **VSAMRS** Same as VM CMS.
CICS for MVS/ESA	If EZERT8 is in the form RSnnnnnn, look under nnnnnn in the return codes section of *VisualAge Generator Server Guide for MVS, VSE, and VM* . Otherwise, the first two characters of EZERT8 contain the hexadecimal representation of the first byte of the EIBFN from the CICS EXEC interface block. The remaining 6 characters contain the hexadecimal representation of bytes 0-2 of the EIBRCODE, also from the CICS EXEC interface block.

Environment	Compatibility considerations
MVS/TSO	If EZERT8 is in the form `RSnnnnnn`, look under `nnnnnn` in the return codes section of *VisualAge Generator Server Guide for MVS, VSE, and VM*

SEQ EZERT8 contains the COBOL status key value or values in the first 2 characters. The remaining 6 characters are zeros.

SEQRS The contents of EZERT8 depend on the operation that failed:

- If a dynamic allocation fails, the first 3 bytes of EZERT8 contain the value S99 (for SVC 99, dynamic allocation), byte 4 is the SVC 99 return code in hexadecimal and bytes 5-8 contain the error reason code in hexadecimal.
- If an OPEN fails, EZERT8 contains return code 8 ('00000008').
- If a READ end-of-file condition occurs, EZERT8 contains return code 4 ('00000004').
- If a READ, WRITE, or CLOSE fails, EZERT8 contains return code 12 ('00000012').

VSAM EZERT8 contains the COBOL status key value or values in the first 2 characters followed by 2 characters for the COBOL VSAM return code (VSAM feedback code), 1 character for the COBOL VSAM function code (VSAM component code), and 3 characters for the COBOL VSAM feedback code (VSAM reason code).

VSAMRS

The operation that fails determines the contents of EZERT8:

- If a dynamic allocation fails, the first 3 bytes of EZERT8 contain the value S99 (for SVC 99, dynamic allocation), byte 4 is the SVC 99 return code in hexadecimal, and bytes 5-8 contain the error reason code in hexadecimal.
- If an OPEN or CLOSE fails, the first 2 bytes of EZERT8 contain the error code from the VSAM program control block (ACB) in hexadecimal. The remaining 6 characters are zeros.
- If an operation other than OPEN or CLOSE fails, the first 2 characters are zeros followed by 2 characters for the COBOL VSAM return code (VSAM feedback code), 1 character for the COBOL VSAM function code (VSAM component code), and 3 characters for the COBOL VSAM feedback code (VSAM reason code).

Environment	Compatibility considerations
MVS batch	If EZERT8 is in the form RSnnnnnn, look under nnnnnn in the return codes section of *VisualAge Generator Server Guide for MVS, VSE, and VM* **GSAM** EZERT8 contains the DL/I status code after an I/O function. The last 6 characters of EZERT8 are spaces. **SEQ** Same as MVS/TSO. **SEQRS** Same as MVS/TSO. **VSAM** Same as MVS/TSO. **VSAMRS** Same as MVS/TSO.
IMS/VS	The only files that can be used in this environment are serial files associated with IMS message queues. EZERT8 contains the DL/I status code after an I/O function to one of these files. The last 6 characters of EZERT8 are spaces.
IMS BMP	**GSAM** EZERT8 contains the DL/I status code after an I/O function. The last 6 characters of EZERT8 are spaces. **IMS message queue** EZERT8 contains the DL/I status code after an I/O function. The last 6 characters of EZERT8 are spaces. Otherwise, same as MVS/TSO.
CICS for VSE/ESA	The first two characters of EZERT8 contain the hexadecimal representation of the first byte of the EIBFN from the CICS EXEC interface block. The third and fourth characters of EZERT8 contain byte 4 of the EIBRCODE, also from the CICS EXEC interface block. The remaining 4 characters of EZERT8 contain the hexadecimal representation of the EIBRESP2 code from the CICS EXEC interface block. EIBRESP2 contains the VSE/POWER return and feedback codes, which can be found in the VSE/POWER PWRDPL copybook.

Environment	Compatibility considerations
VSE batch	If EZERT8 is in the form RSnnnnnn, look under nnnnnn in the return codes section of *VisualAge Generator Server Guide for MVS, VSE, and VM* **SEQ** Same as SEQ in VM CMS. **VSAM** Same as VSAM in VM CMS. **VSAMRS** The operation that fails determines the contents of EZERT8: • If an OPEN or CLOSE fails, the first 2 bytes of EZERT8 contain the error code from the VSAM program control block (ACB) in hexadecimal. The remaining 6 characters are zeros. • If an operation other than OPEN or CLOSE fails, the first 2 characters are zeros followed by 2 characters for the COBOL VSAM return code (VSAM feedback code), 1 character for the COBOL VSAM function code (VSAM component code), and 3 characters for the COBOL VSAM feedback code (VSAM reason code). **SPOOL** The first two characters of EZERT8 contain the hexadecimal value of the VSE/POWER return code (PXPRETCD) from the XPCCB user data area. The following two characters contain the hexadecimal value of the VSE/POWER feedback code (PXPFBKCD) from the XPCCB user data area. The remaining four characters contain zeroes.
CICS for OS/2	For files accessed through COBOL, EZERT8 contains file status key 1 in the first character and file status key 2 in the next 3 characters, followed by 4 characters of zeros. If file status key 1 is the character "9", file status key 2 is converted from single-byte binary to 3-character decimal format. For files accessed through CICS, the first 2 characters of EZERT8 contain the first byte of the EIBFN field from the CICS EXEC interface block. The remaining 6 characters contain bytes 0-2 of the EIBRCODE, also from the CICS EXEC interface block. Otherwise, same as CICS for MVS/ESA.
OS/400	EZERT8 contains the file I/O status code. See "I/O status codes" on page 376.
Windows & OS/2 Smalltalk (GUI)	The first 4 bytes of EZERT8 represent the Client/Server Communication (CSC) subsystem in which the error occurred. The last 4 bytes represent the CSC subsystem error number.
Windows Java (GUI)	Same as Windows & OS/2 Smalltalk (GUI).

Environment	Compatibility considerations
OS/2 (C++)	The /SYSCODES generation option is not supported. The value of EZERT8 is always evaluated. For files accessed through Micro Focus COBOL, EZERT8 contains file status key 1 in the first character, and file status key 2 in the next 3 characters, followed by 4 characters of zeros. If the file status key 1 is the character 9, file status key 2 is converted from single-byte binary to 3-character decimal format. For files accessed through native OS/2 or AIX, the decimal representation of the return code is placed in the low order two bytes of EZERT8. The return code corresponds to the "errno" values (errno.h) that are shipped with the IBM C++ Set compilers.
AIX	Same as OS/2 (C++).
HP-UX	Same as OS/2 (C++).
Solaris	Same as OS/2 (C++).
CICS for Solaris	The /SYSCODES generation option is not supported. The value of EZERT8 is always evaluated. For sequential file access, the decimal representation of the return code is placed in the low order two bytes of EZERT8. The return code corresponds to the "errno" values (errno.h) that are shipped with the IBM C++ Set compilers. For files accessed through CICS, the first two characters of EZERT8 contain the hexadecimal representation of the first byte of the EIBFN field from the CICS EXEC interface block. The remaining six characters contain the hexadecimal representation of bytes 0-2 of the EIBRCODE, also from the CICS EXEC interface block.
CICS for AIX	The /SYSCODES generation option is not supported. The value of EZERT8 is always evaluated. For sequential file access, the decimal representation of the return code is placed in the low order two bytes of EZERT8. The return code corresponds to the "errno" values (errno.h) that are shipped with the IBM C++ Set compilers. For files accessed through CICS, the first two characters of EZERT8 contain the hexadecimal representation of the first byte of the EIBFN field from the CICS EXEC interface block. The remaining six characters contain the hexadecimal representation of bytes 0-2 of the EIBRCODE, also from the CICS EXEC interface block.
Windows NT (C++)	Same as OS/2 (C++).

Environment	Compatibility considerations
Windows NT (Java)	The /SYSCODES generation option is not supported. EZERT8can be filled with a CSO or VGJ message number documented in the (Messages Book).
CICS for Windows NT	Same as CICS for AIX.
Test Facility	EZERT8 can be filled with any of the following: • A VisualAge Generator return code that is documented in the return codes section of the VisualAge Generator Running documents. • A return code from the File Access Method. • An OS/2 system return code. These return codes will be in the form OSnnnnnn. • A CSO message number documented in the *VisualAge Generator Messages and Problem Determination Guide* document.

Example for EZERT8

```
IF EZERT8 = '00000008';
END;
```

EZESEGM

EZESEGM changes CONVERSE processing to either segmented mode or nonsegmented mode.

You can use EZESEGM to specify that a portion of a program is to run in a particular execution mode regardless of how the rest of the program runs.

If EZESEGM is set to 1, the CONVERSE I/O option runs in segmented mode. If EZESEGM is set to 0, the CONVERSE I/O option runs in nonsegmented mode.

The default value of EZESEGM is 0 for nonsegmented programs and 1 for segmented or single-segmented programs. EZESEGM is reset to the default after the CONVERSE I/O option is complete.

EZESEGM is ignored in called programs.

For additional information about segmented programs, refer to *VisualAge Generator Design Guide*.

Uses

You can use EZESEGM as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement

- The target operand in a MOVE or assignment statement
- An occurrence operand in a MOVEA statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

The characteristics of EZESEGM follow:

Data type
Numeric

Data length in bytes
1

Value saved across segments
No

Target environments for EZESEGM

Environment	Compatibility considerations
VM CMS	If EZESEGM is set to 1, segmented mode is simulated by the following: • Committing database changes before each CONVERSE • Refreshing single user table contents • Resetting to their default values the EZE words that are not saved across segments.
VM batch	Ignored.
CICS for MVS/ESA	None.
MVS/TSO	Same as VM CMS.
MVS batch	Ignored.
IMS/VS	Ignored. All programs must be segmented.
IMS BMP	Ignored.
CICS for VSE/ESA	None.
VSE batch	Ignored.
CICS for OS/2	None.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	Same as VM CMS.

Environment	Compatibility considerations
AIX	Same as VM CMS.
HP-UX	Same as VM CMS.
Solaris	Same as VM CMS.
CICS for Solaris	Same as VM CMS.
CICS for AIX	Same as VM CMS.
Windows NT (C++)	Same as VM CMS.
Windows NT (Java)	Same as VM CMS.
CICS for Windows NT	Same as VM CMS.
Test Facility	None.

Example for EZESEGM

```
MOVE 0 TO EZESEGM;
```

EZESEGTR

EZESEGTR changes the name of the transaction code for the next program segment when the program runs. The program stores the transaction name in EZESEGTR prior to a CONVERSE I/O option.

Initially, EZESEGTR contains the segmented transaction name defined when the program is generated. If the program is started using a DXFR statement, EZESEGTR contains the segmented transaction name for the program that initiated the DXFR statement.

The program can change EZESEGTR to another transaction name. When the next segment is started, this new transaction name is used. EZESEGTR remains at its new value in the new segment.

Uses

You can use EZESEGTR as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand in a MOVE or assignment statement
- Data item 1 or 2 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand on the TEST statement

The characteristics of EZESEGTR follow:

Data type
Character

Data length in bytes
8

Value saved across segments
Yes

Target environments for EZESEGTR

Environment	Compatibility considerations
VM CMS	Ignored.
VM batch	Ignored.
CICS for MVS/ESA	If a segmented transaction name is specified at generation, it is used to set the initial contents of EZESEGTR. If a segmented transaction name is not specified at generation, the CICS transaction code is used to set EZESEGTR. The program can change EZESEGTR. The contents of EZESEGTR at a CONVERSE are used to set the next transaction code.
MVS/TSO	Ignored.
MVS batch	Ignored.
IMS/VS	For transaction programs, the initial value of EZESEGTR for the initial program is the IMS transaction code being used when the program is started. For a batch program that scans the I/O PCB, EZESEGTR is reset to the transaction code from the IMS message header each time a SCAN results in a successful get unique call to the I/O PCB.
IMS BMP	If the program runs as a batch-oriented BMP, EZESEGTR is ignored. If the program runs as a transaction-oriented BMP, EZESEGTR is reset to the transaction code from the IMS message header each time a SCAN results in a successful get unique call to the I/O PCB.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	Ignored.
CICS for OS/2	Same as CICS for MVS/ESA.
OS/400	Not supported.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.

Environment	Compatibility considerations
OS/2 (C++)	Ignored. An 8-character data item is allocated and can be used in the various statements, although the item has no effect. Programs in this environment run as nonsegmented transactions.
AIX	Same as OS/2 (C++).
HP-UX	Same as OS/2 (C++).
Solaris	Same as OS/2 (C++).
CICS for Solaris	Same as OS/2 (C++).
CICS for AIX	Same as OS/2 (C++).
Windows NT (C++)	Same as OS/2 (C++).
Windows NT (Java)	Ignored.
CICS for Windows NT	Same as OS/2 (C++).
Test Facility	Ignored during testing.

Example for EZESEGTR

```
MOVE "TRXZ" to EZESEGTR;
```

EZESQCOD (SQL)

EZESQCOD contains the return code for the most recently completed SQL I/O option. It is obtained from the SQL communications area (SQLCA).

Uses

You can use EZESQCOD as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- An occurrence operand in a MOVEA statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

The characteristics of EZESQCOD follow:

Data type
Binary

Data length in bytes
4

Value saved across segments
No

Target environments for EZESQCOD

Environment	Compatibility considerations
VM CMS	The information returned in the SQLCA varies with relational database manager. On this system, the database manager is SQL/DS VM.
VM batch	Same as VM CMS.
CICS for MVS/ESA	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2.
MVS/TSO	Same as CICS for MVS/ESA.
MVS batch	Same as CICS for MVS/ESA.
IMS/VS	Same as CICS for MVS/ESA.
IMS BMP	Same as CICS for MVS/ESA.
CICS for VSE/ESA	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/VSE.
VSE batch	Same as CICS for VSE/ESA.
CICS for OS/2	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/2.
OS/400	The information returned in the SQLCA varies with the relational database manager. In OS/400, the database manager (SQL/400) is included in the base operating system.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/2.
AIX	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
HP-UX	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
Solaris	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
CICS for Solaris	The information returned in the SQLCA varies with the relational database manager.
CICS for AIX	The information returned in the SQLCA varies with the relational database manager.
Windows NT (C++)	Same as OS/2 (C++).

Environment	Compatibility considerations
Windows NT (Java)	Contains the DBMS native return code for the last completed SQL I/O function. For portability, use VAGen I/O Error Values.
CICS for Windows NT	Same as CICS for AIX.
Test Facility	None.

Example for EZESQCOD

```
MOVE EZESQCOD TO RCITEM;
```

EZESQISL (SQL)

EZESQISL contains the isolation level of SQL I/O functions. If set to 0 (the default), the isolation level is set to repeatable read. If set to 1, the isolation level is set to cursor stability.

Note: EZESQISL is supported in this release for upward compatibility, but has no meaning in the supported environments.

Uses

You can use EZESQISL as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand in a MOVE, or assignment statement
- An occurrence operand in a MOVEA statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

The characteristics of EZESQISL follow:

Data type
Numeric

Data length in bytes
1

Value saved across segments
Yes

Definition considerations for EZESQISL

The isolation level for INQUIRY, UPDATE, ADD, SETINQ, SETUPD, and SQLEXEC functions is the isolation level in effect when the function is run.

The isolation level for SCAN, REPLACE, and DELETE functions is the isolation level in effect when the previous UPDATE, SETINQ, or SETUPD for the same SQL row was run.

Target environments for EZESQISL

Environment	Compatibility considerations
VM CMS	Ignored. The isolation level is specified during SQL preprocessing and cannot be dynamically controlled from the program.
VM batch	Same as VM CMS.
CICS for MVS/ESA	Ignored. On DB2 systems, isolation level is specified during BIND processing and cannot be dynamically controlled from the program.
MVS/TSO	Same as CICS for MVS/ESA.
MVS batch	Same as CICS for MVS/ESA.
IMS/VS	Same as CICS for MVS/ESA.
IMS BMP	Same as CICS for MVS/ESA.
CICS for VSE/ESA	Ignored. The isolation level is specified during SQL preprocessing and cannot be dynamically controlled from the program.
VSE batch	Same as CICS for VSE/ESA.
CICS for OS/2	Not supported.
OS/400	Ignored. The isolation level is specified during program CREATE processing with the COMMIT keyword and cannot be dynamically controlled from the program.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	Ignored. On DB2/2 systems, isolation level is specified during BIND processing and cannot be dynamically controlled from the program. On this system, the database manager is DB2/2.
AIX	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
HP-UX	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
Solaris	Ignored.
CICS for Solaris	Ignored.
CICS for AIX	Same as OS/2 (C++).
Windows NT (C++)	Same as OS/2 (C++).
Windows NT (Java)	This value is currently ignored for the JDBC environment. Database default is assumed. For DB2, the default setting is TRANSACTION_READ_COMMITTED.

Environment	Compatibility considerations
CICS for Windows NT	Same as OS/2 (C++).
Test Facility	Not supported.

Example for EZESQISL

```
MOVE 1 TO EZESQISL;
```

EZESQLCA (SQL)

EZESQLCA contains the entire SQL communication area (SQLCA) returned for the last SQL I/O option.

Uses

You can use EZESQLCA as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

In order to refer to specific fields in the SQLCA you will need to move EZESQLCA to a working storage record. The record should have substructure fields defined as specified in the SQLCA description in your DB2 SQL reference manual.

Use the substructured working storage record if you pass the SQLCA contents to a remote program so that the contents will be converted correctly to the remote system data format.

The characteristics of EZESQLCA follow:

Data type
Hexadecimal

Data length in bytes
136

Value saved across segments
No

Target environments for EZESQLCA

Environment	Compatibility considerations
VM CMS	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is SQL/DS VM.

Environment	Compatibility considerations
VM batch	Same as VM CMS.
CICS for MVS/ESA	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2.
MVS/TSO	Same as CICS for MVS/ESA.
MVS batch	Same as CICS for MVS/ESA.
IMS/VS	Same as CICS for MVS/ESA.
IMS BMP	Same as CICS for MVS/ESA.
CICS for VSE/ESA	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/VSE.
VSE batch	Same as CICS for VSE/ESA.
CICS for OS/2	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/2.
OS/400	The information returned in the SQLCA varies with the relational database manager. In OS/400, the database manager (SQL/400) is included in the base operating system.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/2.
AIX	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
HP-UX	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
Solaris	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
CICS for Solaris	The information returned in the SQLCA varies with the relational database manager.
CICS for AIX	The information returned in the SQLCA varies with the relational database manager.
Windows NT (C++)	Same as OS/2 (C++).
Windows NT (Java)	For specific information about fields within EZESQLCA refer to the corresponding EZE words as documented in this section.
CICS for Windows NT	Same as CICS for AIX.

Environment	Compatibility considerations
Test Facility	None.

Example for EZESQLCA

```
MOVE EZESQLCA TO ITEM1;
```

EZESQRD3 (SQL)

EZESQRD3 contains the third integer return value in the SQL communications area (SQLCA) returned for the last SQL I/O option. It returns the number of rows processed for some SQL requests.

Uses

You can use EZESQRD3 as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- An occurrence operand in a MOVEA statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

The characteristics of EZESQRD3 follow:

Data type
Binary

Data length in bytes
4

Value saved across segments
No

Target environments for EZESQRD3

Environment	Compatibility considerations
VM CMS	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is SQL/DS VM.
VM batch	Same as VM CMS.
CICS for MVS/ESA	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2.
MVS/TSO	Same as CICS for MVS/ESA.
MVS batch	Same as CICS for MVS/ESA.
IMS/VS	Same as CICS for MVS/ESA.
IMS BMP	Same as CICS for MVS/ESA.

Environment	Compatibility considerations
CICS for VSE/ESA	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/VSE.
VSE batch	Same as CICS for VSE/ESA.
CICS for OS/2	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/2.
OS/400	The information returned in the SQLCA varies with the relational database manager. In OS/400, the database manager (SQL/400) is included in the base operating system.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/2.
AIX	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
HP-UX	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
Solaris	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
CICS for Solaris	The information returned in the SQLCA varies with the relational database manager.
CICS for AIX	The information returned in the SQLCA varies with the relational database manager.
Windows NT (C++)	Same as OS/2 (C++).
Windows NT (Java)	Currently has no meaning for the JDBC environment. It will not be updated as the program executes, and it will always have its default value unless explicitly changed by the program.
CICS for Windows NT	Same as CICS for AIX.
Test Facility	None.

Example for EZESQRD3

```
MOVE EZESQRD3 TO RCITEM2;
```

EZESQRRM (SQL)

EZESQRRM contains the substitution variables for the error message associated with the return code in EZESQCOD. EZESQRRM is obtained from the SQL communications area (SQLCA) for the last SQL I/O option.

Uses

You can use EZESQRRM as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand on the TEST statement

The characteristics of EZESQRRM follow:

Data type
Character

Data length in bytes
70

Value saved across segments
No

Definition considerations for EZESQRRM

The substitution variables are separated by a single byte containing X'FF'.

Target environments for EZESQRRM

Environment	Compatibility considerations
VM CMS	The information returned in the SQLCA varies with the relational database manager. On this systems, the database manager is SQL/DS VM.
VM batch	Same as VM CMS.
CICS for MVS/ESA	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2.
MVS/TSO	Same as CICS for MVS/ESA.
MVS batch	Same as CICS for MVS/ESA.
IMS/VS	Same as CICS for MVS/ESA.
IMS BMP	Same as CICS for MVS/ESA.
CICS for VSE/ESA	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/VSE.
VSE batch	Same as CICS for VSE/ESA.

Environment	Compatibility considerations
CICS for OS/2	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/2.
OS/400	The information returned in the SQLCA varies with the relational database manager. The character string is formatted for OS/400 messages, and does not contain the X'FF' separators. In OS/400, the database manager (SQL/400) is included in the base operating system.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/2.
AIX	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
HP-UX	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
Solaris	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
CICS for Solaris	The information returned in the SQLCA varies with the relational database manager.
CICS for AIX	The information returned in the SQLCA varies with the relational database manager.
Windows NT (C++)	Same as OS/2 (C++).
Windows NT (Java)	Currently has no meaning for the JDBC environment. It will not be updated as the program executes, and it will always have its default value unless explicitly changed by the program.
CICS for Windows NT	Same as CICS for AIX.
Test Facility	None.

Example for EZESQRRM

```
MOVE EZESQRRM TO RCITEM3;
```

EZESQWN1 (SQL)

EZESQWN1 is the second warning byte returned in the SQL communications area (SQLCA) for the last SQL I/O option. EZESQWN1 indicates whether character data items were truncated.

Uses

You can use EZESQWN1 as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand in a TEST statement

The characteristics of EZESQWN1 follow:

Data type
Character

Data length in bytes
1

Value saved across segments
No

Definition considerations for EZESQWN1

EZESQWN1 contains a W if the last SQL I/O option caused the database manager to truncate character data items because of insufficient space in the program host variables. You can test specific fields to determine which ones were truncated by using the following:
- TEST item TRUNC
- IF item IS TRUNC
- IF item NOT TRUNC
- WHILE item IS TRUNC
- WHILE item NOT TRUNC

When the data item is a number, no truncation warning is given. Fractional parts of a number will be truncated with no indication. If the non-fractional part of a number will not fit into a user variable, the database manager returns a -304 in EZESQCOD when DB2 is used.

Target environments for EZESQWN1

Environment	Compatibility considerations
VM CMS	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is SQL/DS VM.
VM batch	Same as VM CMS.
CICS for MVS/ESA	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2.
MVS/TSO	Same as CICS for MVS/ESA.
MVS batch	Same as CICS for MVS/ESA.

Environment	Compatibility considerations
IMS/VS	Same as CICS for MVS/ESA.
IMS BMP	Same as CICS for MVS/ESA.
CICS for VSE/ESA	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/VSE.
VSE batch	Same as CICS for VSE/ESA.
CICS for OS/2	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/2.
OS/400	The information returned in the SQLCA varies with the relational database manager. In OS/400, the database manager (SQL/400) is included in the base operating system.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/2.
AIX	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
HP-UX	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
Solaris	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
CICS for Solaris	The information returned in the SQLCA varies with the relational database manager.
CICS for AIX	The information returned in the SQLCA varies with the relational database manager.
Windows NT (C++)	Same as OS/2 (C++).
Windows NT (Java)	None.
CICS for Windows NT	Same as CICS for AIX.
Test Facility	None.

Example for EZESQWN1

In the following example, MY-CHAR-FIELD is a field in the SQL row record just processed and LOST-DATA is a function that sets an error message indicating that information for MY-CHAR-FIELD was truncated.

```
IF EZESQWN1 = 'W';
  TEST MY-CHAR-FIELD TRUNC LOST-DATA;
END;
```

EZESQWN6 (SQL)

EZESQWN6 is the seventh warning byte returned in the SQL communications area (SQLCA) for the last SQL I/O option. The meaning of the EZESQWN6 field is dependent on the database manager.

Uses

You can use EZESQWN6 as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand in a TEST statement

The characteristics of EZESQWN6 follow:

Data type
Character

Data length in bytes
1

Value saved across segments
No

Definition considerations for EZESQWN6

For DB2/VSE, EZESQWN6 contains a W or an S if the last SQL statement processed caused SQL to back out all requests from this program since the last call to EZECOMIT. This field contains an S if the last call to DB2/VSE caused an error so severe that any further attempt to communicate with SQL would cause the program to end.

For DB2, EZESQWN6 contains a W when an adjustment was made to correct a result that was not valid from an arithmetic operation on date or time values.

For DB2/2 Version 1.0, EZESQWN6 contains a W if the result of a date calculation was adjusted to avoid an impossible date.

S is not valid on workstation relational database management systems or DB2.

Target environments for EZESQWN6

Environment	Compatibility considerations
VM CMS	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is SQL/DS VM.
VM batch	Same as VM CMS.
CICS for MVS/ESA	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2.
MVS/TSO	Same as CICS for MVS/ESA.
MVS batch	Same as CICS for MVS/ESA.
IMS/VS	Same as CICS for MVS/ESA.
IMS BMP	Same as CICS for MVS/ESA.
CICS for VSE/ESA	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/VSE.
VSE batch	Same as CICS for VSE/ESA.
CICS for OS/2	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/2.
OS/400	The information returned in the SQLCA varies with the relational database manager. In OS/400 the database manager (SQL/400) is included in the base operating system.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	The information returned in the SQLCA varies with the relational database manager. On this system, the database manager is DB2/2.
AIX	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
HP-UX	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
Solaris	Same as OS/2 (C++). On this system, the database manager is DB2/6000.
CICS for Solaris	The information returned in the SQLCA varies with the relational database manager.
CICS for AIX	The information returned in the SQLCA varies with the relational database manager.
Windows NT (C++)	Same as OS/2 (C++).

Environment	Compatibility considerations
Windows NT (Java)	Currently has no meaning for the JDBC invironment. It will not be updated as the program executes, and it will always have its default value unless explicitly changed by the program.
CICS for Windows NT	Same as CICS for AIX.
Test Facility	None.

Example for EZESQWN6

```
IF EZESQWN6 = 'S';
```

EZESYS

EZESYS identifies the environment in which the program is running.

The following are valid values for EZESYS:
- AIX
- AIXCICS
- HP
- IMSBMP
- IMSVS
- ITF
- MVSBATCH
- MVSCICS
- NTCICS
- OS2
- OS2CICS
- OS2GUI
- OS400
- SOLARIS
- SOLACICS
- TSO
- VMCMS
- VMBATCH
- VSEBATCH
- VSECICS
- WINGUI
- WINNT

Uses

You can use EZESYS as any of the following:
- The source operand in an assignment, MOVE, or MOVEA statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

- An operand on the TEST statement

The characteristics of EZESYS follow:

Data type
Character

Data length in bytes
8 (padded with spaces)

Value saved across segments
Yes

Definition considerations for EZESYS

EZESYS tests are implemented as run-time tests, not generation tests. Therefore, you cannot use EZESYS in conjunction with the IF, WHILE, or TEST statements to optionally include or exclude functions in generated programs.

Consider the following sequence of code:

```
IF EZESYS NOT IMSVS;
   MY_VSAM_FUNC();     /* Perform my VSAM function */
END;
```

Because VSAM is not supported for the IMS/VS environment, you cannot generate the program for IMS/VS. To generate the program for IMS/VS, you must move the VSAM functions to another program and change the IF statement to the following:

```
IF EZESYS NOT IMSVS;
  CALL VSAMAPP VSAM_FUNC,RECORD;
END;
```

Similar measures must be taken to optionally execute any function that will generate for one environment, but not for another.

Target environments for EZESYS

Supported in all environments without compatibility considerations.

Examples for EZESYS

The following is an CICS for OS/2 example:

```
IF EZESYS IS OS2CICS;
END;
```

The following is an OS/2 (GUI) and Windows example:

```
IF EZESYS IS WINGUI;
  MOVE "Windows" TO ENVRECD.UI;
ELSE;
  IF EZESYS IS OS2GUI;
    MOVE "OS/2" TO ENVRECD.UI;
```

```
  ELSE;
    MOVE "NPT" TO ENVRECD.UI;
  END;
END;
```

The "IF EZESYS IS WINGUI;" statement above could be coded in a function in a GUI program or a non-GUI program, and could be generated and run in any target environment. That is, EZESYS could be tested for being WINGUI anywhere, but would only test "true" in a GUI program running under Windows, and would test "false" everywhere else.

EZETIM

EZETIM contains the current system time in the format of HH:MM:SS. EZETIM is automatically updated each time your program refers to it.

Uses

You can use EZETIM as the source operand in a MOVE, MOVEA, or assignment statement.

The characteristics of EZETIM follow:

Data type
Character

Data length in bytes
8

Value saved across segments
No

Target environments for EZETIM

Supported in all environments without compatibility considerations.

Example for EZETIM

```
MOVE EZETIM TO TIMFLD;
```

EZETST

EZETST contains the following:

- The number of the first row in the table that meets the search conditions specified in a FIND or RETR statement.
- The number of the first element in an array that matches the search conditions in an IF or WHILE statement with an IN operator.
- The subscript of the last element modified in the target array after a MOVEA statement.

If a row or an array element is not found, EZETST contains 0. Once EZETST is set, you can specify it as a table or array subscript to access the data in the selected row or array element.

Uses

You can use EZETST as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- The target operand in a MOVE or assignment statement
- An occurrence operand in a MOVEA statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement

The characteristics of EZETST follow:

Data type
Binary

Data length in bytes
2

Value saved across segments
No

Target environments for EZETST

Supported in all environments without compatibility considerations.

Example for EZETST

If you have a table named INFO with 50 rows and 3 columns named STATE, AREA, and POPULATION, you could use the following sequence to set the variable PEOPLE to the population for Alaska. AMOUNT is set with the RETR statement to the area.

In the following table, the first column has an entry for each of the 50 states, the second column contains the population for each state, and the third column contains the area in square miles for each state.

INFO:

```
    STATE      POPULATION       AREA

1   Alabama    3,500,000        51,600
2   Alaska       302,000       586,000
        .             .            .
        .             .            .
        .             .            .
```

A RETR statement could be used to pick up the area information from the above table, based on a matching state.

```
MOVE 'ALASKA' TO ITEM;
RETR ITEM INFO.STATE AMOUNT AREA;
```

AMOUNT now has 586,000 in it. EZETST contains 2, the row number of the matching state. If no match is made, EZETST is set to 0 and the contents of AMOUNT are not changed. If the match is found, you can now obtain the population for ALASKA by the following statement:

```
MOVE INFO.POPULATION[EZETST] TO PEOPLE;
```

You could also code:

```
PEOPLE = INFO.POPULATION[EZETST];
```

EZEUSR

EZEUSR contains the system-dependent user identifier or terminal identifier for your program.

EZEUSR is supported only for compatibility with releases previous to CSP 370AD Version 4 Release 1.

Note: For Web Transaction programs, EZEUSR contains the user connection id code. Use the EZEUSR value as a key value to access file or database information shared between transactions running on behalf of a single web user within a single Internet session.

Uses

You can use EZEUSR as any of the following:

- The source operand in a MOVE, MOVEA, or assignment statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand on the TEST statement

The characteristics of EZEUSR follow:

Data type
Character

Data length in bytes
8

Value saved across segments
Yes

Target environments for EZEUSR

Environment	Compatibility considerations
VM CMS	EZEUSR contains the VM logon user ID. EZEUSR is equivalent to EZEUSRID.
VM batch	EZEUSR contains the VM userid of the VM batch virtual machine running the program. EZEUSR is equivalent to EZEUSRID.
CICS for MVS/ESA	EZEUSR contains the CICS terminal identifier. EZEUSR is equivalent to EZELTERM. See note on Web Transaction programs at the beginning of the EZEUSR section.
MVS/TSO	EZEUSR contains the TSO logon identifier. EZEUSR is equivalent to EZEUSRID.
MVS batch	EZEUSR contains the job name from the JOB card. EZEUSR is equivalent to EZEUSRID.
IMS/VS	EZEUSR contains the user ID field from I/O PCB. EZEUSR is updated whenever there is a successful get unique call to the I/O PCB. This is caused by a SCAN for a serial file associated with the I/O PCB, a CONVERSE I/O option, or a first map. EZEUSR is set to spaces when a main batch program that scans the message queue gets an EOF (GC status code for a get unique call). The user ID field is blank if sign-on security is not active on the system. EZEUSR is equivalent to EZEUSRID. See note on Web Transaction programs at the beginning of the EZEUSR section.
IMS BMP	If the program runs as an IMS batch-oriented BMP, EZEUSR is set to the job name from the JOB card. If the program runs as an IMS transaction-oriented BMP, EZEUSR is initialized to the name of the job from the JOB card of the JCL. EZEUSR is reset to the user ID field from the I/O PCB on each SCAN that results in a successful get unique call for a serial file associated with the I/O PCB. The user ID field is blank if sign-on security is not active on the system. EZEUSR is equivalent to EZEUSRID.

Environment	Compatibility considerations
CICS for VSE/ESA	Same as CICS for MVS/ESA. See note on Web Transaction programs at the beginning of the EZEUSR section.
VSE batch	Same as MVS batch.
CICS for OS/2	Same as CICS for MVS/ESA. See note on Web Transaction programs at the beginning of the EZEUSR section.
OS/400	None.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	EZEUSR contains “PC user”, for compatibility with the VisualAge Generator Test Facility. See note on Web Transaction programs at the beginning of the EZEUSR section.
AIX	EZEUSR contains the first 8 characters of the logon identifier. See note on Web Transaction programs at the beginning of the EZEUSR section.
HP-UX	EZEUSR contains the first 8 characters of the logon identifier. See note on Web Transaction programs at the beginning of the EZEUSR section.
Solaris	EZEUSR contains the first 8 characters of the logon identifier. See note on Web Transaction programs at the beginning of the EZEUSR section.
CICS for Solaris	Same as CICS for MVS/ESA. See note on Web Transaction programs at the beginning of the EZEUSR section.
CICS for AIX	Same as CICS for MVS/ESA. See note on Web Transaction programs at the beginning of the EZEUSR section.
Windows NT (C++)	Same as OS/2 (C++). See note on Web Transaction programs at the beginning of the EZEUSR section.

Environment	Compatibility considerations
Windows NT (Java)	In main and called programs, EZEUSR is initialized from the system property user.name. If the property cannot be retrieved, EZEUSR is blank.
CICS for Windows NT	Same as CICS for MVS/ESA. See note on Web Transaction programs at the beginning of the EZEUSR section.
Test Facility	EZEUSR contains “PC user”.

Example for EZEUSR

```
MOVE EZEUSR to ITEM7;
```

EZEUSRID

EZEUSRID contains a user identifier in environments that have a user ID available.

Uses

You can use EZEUSRID as any of the following:
- The source operand in a MOVE, MOVEA, or assignment statement
- Data item 1 in a RETR statement
- A data item in an IF or WHILE statement
- A data item in a FIND statement
- An operand on the TEST statement

The characteristics of EZEUSRID follow:

Data type
Character

Data length in bytes
8 (padded with spaces)

Value saved across segments
Yes

Target environments for EZEUSRID

Environment	Compatibility considerations
VM CMS	EZEUSRID contains the VM logon user ID. EZEUSRID is equivalent to EZEUSR.
VM batch	EZEUSRID contains the VM userid of the VM batch virtual machine running the program. EZEUSRID is equivalent to EZEUSR.

Environment	Compatibility considerations
CICS for MVS/ESA	EZEUSRID contains the CICS user ID. If the user signed onto the system, EZEUSRID contains the user ID specified at sign-on. When RACF is installed, this is the RACF user ID. If the user did not sign on, EZEUSRID contains spaces.
MVS/TSO	EZEUSRID contains the TSO logon identifier. EZEUSRID is equivalent to EZEUSR.
MVS batch	EZEUSRID contains the job name from the JOB card. EZEUSRID is equivalent to EZEUSR.
IMS/VS	EZEUSRID contains the user ID field from the I/O PCB. EZEUSRID is updated whenever there is a successful get unique call to the I/O PCB. This is caused by a SCAN for a serial file associated with the I/O PCB, a CONVERSE I/O option, or a first map. EZEUSRID is set to spaces when a main batch program that scans the message queue gets an EOF (GC status code for a get unique call). The user ID field is blank if sign-on security is not active on the system. EZEUSRID is equivalent to EZEUSR.
IMS BMP	If the program runs as an IMS batch-oriented BMP, EZEUSRID contains the job name from the JOB card. If the program runs as an IMS transaction-oriented BMP, EZEUSRID is initialized to the name of the job from the JOB card of the JCL. It is updated with the user ID field from the I/O PCB on each SCAN that results in a successful get unique call for a serial file associated with the I/O PCB. The user ID field is blank if sign-on security is not active on the system. EZEUSRID is equivalent to EZEUSR.
CICS for VSE/ESA	If the program user is signed on to the VSE Interactive User Interface (IUI), EZEUSRID contains the user ID specified at sign-on. If the user did not sign on, EZEUSRID contains spaces.
VSE batch	Same as MVS batch.
CICS for OS/2	EZEUSRID contains the CICS user ID. If the user signed onto the system, EZEUSRID contains the user ID specified at sign-on. If the user did not sign on, EZEUSRID contains spaces.
OS/400	None.

Environment	Compatibility considerations
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	EZEUSRID contains spaces for compatibility with the VisualAge Generator Test Facility.
AIX	EZEUSRID contains the first 8 characters of the logon identifier.
HP-UX	EZEUSRID contains the first 8 characters of the logon identifier.
Solaris	EZEUSRID contains the first 8 characters of the logon identifier.
CICS for Solaris	EZEUSRID contains the CICS user ID.
CICS for AIX	EZEUSRID contains the CICS user ID.
Windows NT (C++)	Same as OS/2 (C++).
Windows NT (Java)	EZEUSRID is initialized from the system property user.name. If the property cannot be retrieved, EZEUSRID is blank.
CICS for Windows NT	Same as CICS for AIX.
Test Facility	EZEUSRID is blank by default but may be specified in the VAGen Options/Preferences menu.

Example for EZEUSRID

```
MOVE EZEUSRID TO ITEM12;
```

EZEWAIT

EZEWAIT enables a program to suspend activities for a specified amount of time.

Uses

You can use EZEWAIT as the function name in a function invocation statement.

The calling sequence for EZEWAIT is:

time
A numeric data item specifying the waiting period in hundredths of seconds.

You can use EZEWAIT when two asynchronously running programs need to communicate through a record in a shared file or database. One program might need to suspend processing, without tying up computer system resources, until the other program updates the information in the shared record.

Target environments for EZEWAIT

Environment	Compatibility considerations
VM CMS	None.
VM batch	None.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	None.
IMS/VS	Ignored.
IMS BMP	None.
CICS for VSE/ESA	None.
VSE batch	None.
CICS for OS/2	None.
OS/400	None.
Windows & OS/2 Smalltalk (GUI)	Not supported.
Windows Java (GUI)	Not supported.
OS/2 (C++)	None.
AIX	None.
HP-UX	None.
Solaris	None.
CICS for Solaris	None.
CICS for AIX	None.
Windows NT (C++)	None.
Windows NT (Java)	None.
CICS for Windows NT	None.

Environment	Compatibility considerations
Test Facility	Not supported.

Example for EZEWAIT

The following will cause the program to wait for 15 seconds:

```
MOVE 1500 TO WAIT-TIME;
EZEWAIT(WAIT-TIME);
```

Chapter 12. String function words

String function words are reserved names in VisualAge Generator. String function words provide string-handling capabilities to the VisualAge Generator language. Using string functions, a program can perform operations such as copies, comparisons and concatenations on substrings within string data items.

A string is a fixed-length sequence of bytes. For VisualAge Generator, a string is a data item with a type of CHA, MIX, DBCS, HEX, or UNICODE. A substring is a subset of a string identified by an index and a length. The index value identifies the starting byte of the substring within the data item. The index value for the first byte in the item is 1. The length is the number of bytes in the substring.

To prevent substring definition from extending outside a data item, the index must be a value between 1 and the number of bytes in the item, and the substring length must not extend beyond the end of the data item that contains the substring. Lengths that are too long are adjusted so that the substring ends at the last byte of the data item.

String functions are invoked using the specified syntax. Arguments must be specified for each function parameter. The test facility and generators check that the arguments are compatible with the parameter definitions.

String function arguments can be record data items at any level, or character or numeric literals, where appropriate. Item names can be qualified or subscripted.

Note: Where numeric literals are valid as string function arguments, you must surround the numeric literal with at least one blank.

Example:

```
func (a, 10.2 ,c);
```

Functions can raise exception conditions. EZEREPLY specifies whether exception codes raised by a function will be returned in EZERT8. EZEREPLY is effective only for functions supplied by VisualAge Generator (VAGen).

The definition of the term ″blank″ in the string function descriptions varies with the data type as listed in Table 23 on page 610.

Table 23. Data types and blank definition

Data item type	Blank value
CHA, MIX	Single byte space character
DBCS	Double byte space character
UNICODE	Uncode blank (X'0020') character
HEX	Null (X'00') character

String function words

Table 24. String function words

Element	COBOL											GUI		C++								
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	Test Facility
EZESBLKT	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZESCCWS	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZESCMPR	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZESCNCT	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZESCOPY	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZESFIND	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZESNULT	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZESSET	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZESTLEN	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZESTOKN	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Note: *Includes Windows 95 and Windows 98, Windows NT, and Windows 2000.

Legend: In this table, the following characters are used to indicate the level of support:
x Supported
c Supported with compatibility considerations
blank Not supported

EZESBLKT

EZESBLKT changes the null terminator and any subsequent characters in a string to spaces.

EZESBLKT changes a null-terminated string value returned from a C program to a character data item value that can operate correctly within a VisualAge Generator program.

Attribute	Description
target	The target string item. The target string is searched for a null terminator. If one is found, the null terminator and any characters following it are changed to spaces.

Target environmentsfor EZESBLKT

Supported in all environments without compatibility considerations.

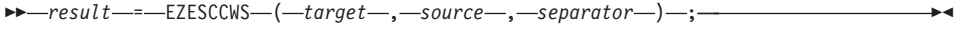
Examplefor EZESBLKT

```
EZESBLKT (TARGET);
```

EZESCCWS

EZESCCWS concatenates one string to another with a separator string between them.

If the initial length of the target string is zero, not counting trailing blanks and nulls, the separator is omitted and only the source string is moved to the target string.



Attribute	Description
result	A numeric data item that will contain the result value returned by the function. The result is a nine-digit binary item. The following values are returned: **0** Concatenated string fits in the target item. **–1** Concatenated string was longer than the target item; characters other than nulls and spaces were truncated from the result.

Attribute	Description
target	The target string item.
source	The source string item or literal.
separator	The separator string.

Trailing spaces and nulls are truncated from the target string. The source and separator strings are appended to the truncated value. The result is truncated to the length of the target string (if the result is longer than the target value) or padded with spaces (if the result is shorter than the target value).

Target environmentsfor EZESCCWS

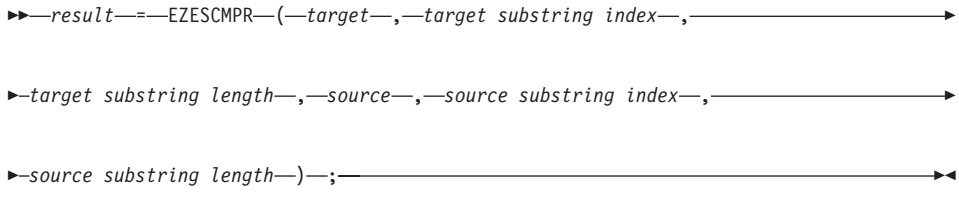
Supported in all environments without compatibility considerations.

Example for EZESCCWS

```
MOVE 'CLIENT NAME =' TO PRINT_LINE;
RESULT = EZESCCWS(PRINT_LINE,CLIENT_NAME,' ');
IF RESULT = -1;
   CALL PRINT_LINE_OVERFLOW;
END;
```

EZESCMPR

EZESCMPR compares one substring to another.



Attribute	Description
result	A numeric data item that will contain the result value returned by the function. The result is a nine-digit binary item. The following values are returned: **1** Target substring is greater than the source substring. **0** Target substring is equal to the source substring. **–1** Target substring is less than the source substring.
target	The target string item.
target substring index	Substring index identifying the starting byte of the target substring within the target item. The substring index is a nine-digit binary item. The index value for the first byte in the string item is 1.

Attribute	Description
target substring length	Target substring length in bytes. The substring length is a nine-digit binary item.
source	The source string item or literal.
source substring index	Substring index identifying the starting byte of the source substring within the source item. The substring index is a nine-digit binary item. The index value for the first byte in the string item is 1.
source substring length	Source substring length in bytes. The substring length is a nine-digit binary item.

The comparison is a byte-by-byte binary comparison of the substring values. If the substrings are not the same length, the shorter substring is padded with spaces prior to the comparison.

Definition considerations for EZESCMPR

The following exception code values are returned:

8 Index less than 1 or greater than string length

12 Length less than 1

20 Invalid double byte index; an index for a DBCS or UNICODE string points to middle of double byte character

24 Invalid double byte length ; length in bytes for DBCS or UNICODE string is odd; length for a double byte string must be even

Numeric literals can be specified in the length and index arguments.

Target environments for EZESCMPR

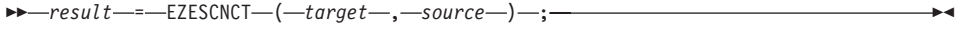
Supported in all environments without compatibility considerations.

Example for EZESCMPR

```
EZEREPLY = 1;
RESULT = EZESCMPR(TARGET,3,2,SOURCE,8,2);
```

EZESCNCT

EZESCNCT concatenates one string to another.



EZESCNCT

Attribute	Description
result	A numeric data item that will contain the result value returned by the function. The result is a nine-digit binary item. The following values are returned: **0** Concatenated string fits in the target item. **–1** Concatenated string was longer than the target item; characters other than nulls and spaces were truncated from the result.
target	The target string item.
source	The source string item or literal.

Trailing spaces and nulls are truncated from the target string. The source and separator strings are appended to the truncated value. The result is truncated to the length of the target string (if the result is longer than the target value) or padded with spaces (if the result is shorter than the target value).

Target environments for EZESCNCT

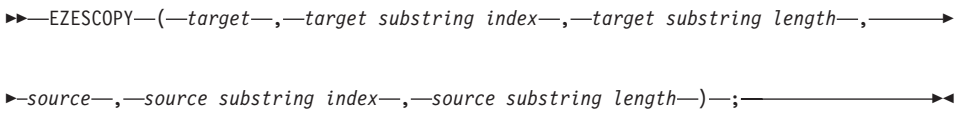
Supported in all environments without compatibility considerations.

Example for EZESCNCT

```
MOVE 'SALARY = $' TO PRINT_LINE;
RESULT = EZESCNCT(PRINT_LINE,SALARY);
IF RESULT = -1;
   CALL PRINT_LINE_OVERFLOW;
END;
```

EZESCOPY

EZESCOPY copies one substring to another.



Attribute	Description
target	The target string item.
target substring index	Substring index identifying the starting byte of the target substring within the target item. The substring index is a nine-digit binary item. The index value for the first byte in the string item is 1.
target substring length	Target substring length in bytes. The substring length is a nine-digit binary item.

Attribute	Description
source	The source string item or literal.
source substring index	Substring index identifying the starting byte of the source substring within the source item. The substring index is a nine-digit binary item. The index value for the first byte in the string item is 1.
source substring length	Source substring length in bytes. The substring length is a nine-digit binary item.

If the source substring is longer than the target substring, the source substring value is truncated when copied to the target substring. If the source substring is shorter than the target substring, the source value is padded on the right with spaces when copied to the target.

Definition considerations for EZESCOPY

The following exception code values are returned:

8 Index less than 1 or greater than string length

12 Length less than 1

20 Invalid double byte index; an index for a DBCS or UNICODE string points to middle of double byte character

24 Invalid double byte length ; length in bytes for DBCS or UNICODE string is odd; length for a double byte string must be even

Numeric literals can be specified in the length and index arguments.

Target environments for EZESCOPY

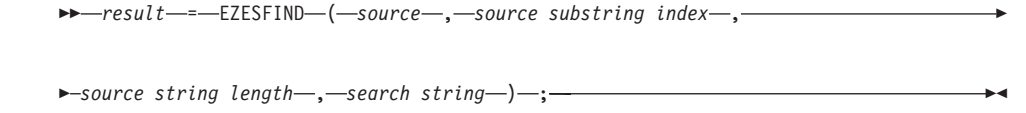
Supported in all environments without compatibility considerations.

Example for EZESCOPY

```
EZEREPLY = 1;
EZESCOPY(COPY_TO,3,6,COPY_FROM,8,6);
```

EZESFIND

EZESFIND finds the first occurrence of a specified string within a string.



Attribute	Description
result	A numeric data item that will contain the result value returned by the function. The result is a nine-digit binary item. The following values are returned: **0** The search string is found. **–1** The search string is not found.
source	The source string item or literal.
source substring index	Substring index identifying the starting byte of the search string within the source item. The substring index is a nine-digit binary item. The index value for the first byte in the string item is 1.
source string length	Source string length in bytes. The string length is a nine-digit binary item.
search string	The search string item or literal. Trailing blanks and nulls are truncated from the search string prior to performing the search.

If the search string is found, the source index is set to the index of the starting byte of the matching substring. Otherwise, the substring index is not changed.

Definition considerations for EZESFIND

The following exception code values are returned:

8 Index less than 1 or greater than string length

12 Length less than 1

20 Invalid double byte index; an index for a DBCS or UNICODE string points to middle of double byte character

24 Invalid double byte length ; length in bytes for DBCS or UNICODE string is odd; length for a double byte string must be even

Numeric literals can be specified in the length argument.

Target environments for EZESFIND

Supported in all environments without compatibility considerations.

Example for EZESFIND

```
INDEX=1;
EZEREPY = 1;
RESULT = EZESFIND(CLIENT_ADDRESS,INDEX,100,SEARCH_ZIP_CODE);
```

EZESNULT

EZESNULT changes trailing spaces to nulls in a string. You can use EZESNULT to change a data item to an argument that can be passed to a C program expecting a null-terminated string.

Attribute	Description
target	The target string item.

The target string is searched for trailing spaces and nulls. Any spaces found are changed to nulls. If no trailing spaces or nulls are found, an exception code is returned in EZERT8.

Definition considerations for EZESNULT

The following exception code value is returned:

16 The last byte of the target string is not a space or null character.

Target environments for EZESNULT

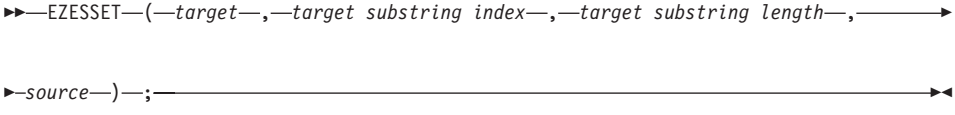
Supported in all environments without compatibility considerations.

Example for EZESNULT

```
EZEREPLY = 1;
EZESNULT(TARGET);
```

EZESSET

EZESSET sets each character in a substring to the same character value.



Attribute	Description
target	The target string item.
target substring index	Substring index identifying the starting byte of the target substring within the target item. The substring index is a nine-digit binary item. The index value for the first byte in the string item is 1.
target substring length	Target substring length in bytes. The substring length is a nine-digit binary item.
source	If the target item is CHA, MIX, or HEX, the character item must be one-byte CHA, MIX, or HEX item or CHA literal. If the target is a DBCS or UNICODE item, the character must be a single character DBCS or UNICODE item.

EZESSET

Definition considerations for EZESSET

The following exception code values are returned:

8 Index less than 1 or greater than string length

12 Length less than 1

20 Invalid double byte index; an index for a DBCS or UNICODE string points to middle of double byte character

24 Invalid double byte length ; length in bytes for DBCS or UNICODE string is odd; length for a double byte string must be even

Numeric literals can be specified in the length and index arguments.

Target environments for EZESSET

Supported in all environments without compatibility considerations.

Example for EZESSET

```
EZEREPLY = 0;
EZESSET(TARGET,12,5,' ');
```

EZESTLEN

EZESTLEN returns the length of an item less trailing spaces and nulls.

Attribute	Description
length	A numeric data item that will contain the returned length of the source string in bytes. Trailing spaces and nulls are not included in the value returned. The result is a nine-digit binary item.
source	The source string item or literal.

Target environments for EZESTLEN

Supported in all environments without compatibility considerations.

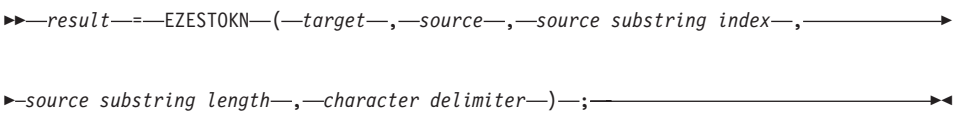
Example for EZESTLEN

```
LENGTH = EZESTLEN(SOURCE);
```

EZESTOKN

EZESTOKN finds the next token in a string and copies the token to an item.

Tokens are substrings separated by delimiter characters. For example, given the string `'CALL PROGRAM ARG1,ARG2, ARG3 '` and the delimiter characters space (' ') and comma (','), the tokens are `'CALL'`, `'PROGRAM'`, `'ARG1'`, `'ARG2'`, and `'ARG3 '`.



Attribute	Description
result	A numeric data item that will contain the result value returned by the function. The result is a four-byte binary item. The following values are returned: **+n** Length of token copied to target item. **0** No token in source substring. **–1** Token truncated when copied to target item.
target	The target string item.
source	The source string item or literal.
source substring index	Substring index identifying the starting byte within the source item at which the search should begin for the delimiting characters. The substring index is a nine-digit binary item. The index value for the first byte in the string item is 1. When a match is found, the value in the substring index is changed to the index of the character following the matched token.
source substring length	Source substring length in bytes. The substring length is a nine-digit binary item. When the call is successful, the value in the source substring length is changed to the length of the substring left following the token that was returned.
character delimiter	The item containing the delimiter characters. If the delimiter item is DBCS or UNICODE, the delimiter characters are double byte characters.

Because the source index and source length values are updated on each successful call, a sequence of calls can be made to retrieve, in order, each of the tokens in a substring, without resetting the source index and source length before each call.

Definition considerations for EZESTOKN

The following exception code values are returned:

8 Index less than 1 or greater than string length

12 Length less than 0

20 Invalid double byte index; an index for a DBCS or UNICODE string points to middle of double byte character

24 Invalid double byte length ; length in bytes for DBCS or UNICODE string is odd; length for a double byte string must be even

Target environments for EZESTOKN

Supported in all environments without compatibility considerations.

Example for EZESTOKN

```
ARGLEN = EZESTOKN(TOKEN,INPUT_LINE,NEXT,REMAINING_LEN,',');
IF TOKEN = 'CALL';
   ARGLEN = EZESTOKN(PROGRAM_NAME,INPUT_LINE,NEXT,REMAINING_LEN,',');
   ARGCOUNT = 0;
   WHILE ARGLEN > 0;
      ARGCOUNT = ARGCOUNT + 1;
      ARGLEN = EZESTOKN(ARG[ARGCOUNT],INPUT_LINE,NEXT,REMAINING_LEN,',');
   END;
   ARGCOUNT = ARGCOUNT - 1;
END;
```

Chapter 13. Math function words

Math function words are reserved names in VisualAge Generator that perform basic mathematical functions, including:

- Transformation between VisualAge Generator numeric data types and floating point numbers
- Floating point arithmetic
- Log functions
- Powers and square root
- Trigonometric functions

Except where noted otherwise, the input parameters are converted to double precision floating point numbers in the format appropriate for the machine on which the program is running. The operation is performed using a C language function and the result is converted back to the format of result parameter.

The term ″numeric data item″ refers to any of the following:

- Any data item with the type NUM, NUMC, PACK, PACF, or BIN.
- A 4-byte HEX item. The item is assumed to be a single precision, 4-byte floating point number native to the run-time environment.
- An 8-byte HEX item. The item is assumed to be a double precision, 8-byte floating point number native to the run-time environment.

Math function exceptions

Functions can raise exception conditions. EZEREPLY specifies whether exception codes raised by a function will be returned in EZERT8. EZEREPLY is effective only for functions supplied by VisualAge Generator (VAGen).

The following exception codes are returned from the math functions:

8 Domain error; argument is not in a valid range for the function to operate on

12 Range error; intermediate or final result cannot be represented as a double precision floating point number, or with the precision of the result parameter.

16 C math function exception

Math function words

Table 25. Math function words

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
EZEABS	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEACOS	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEASIN	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEATAN	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEATAN2	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZECEIL	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZECOS	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZECOSH	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEEXP	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEFLADD	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEFLDIV	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEFLMOD	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEFLMUL	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEFLOOR	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEFLSET	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEFLSUB	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEFREXP	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZELDEXP	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZELOG	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZELOG10	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEMAX	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEMIN	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEMODF	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZENCMPR	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEPOW	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Table 25. Math function words (continued)

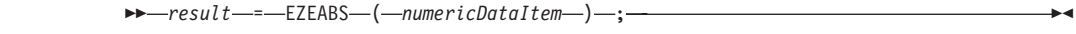
Element	COBOL											GUI		C++								
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	Test Facility
EZEPRCSN	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZEROUND	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZESIN	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZESINH	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZESQRT	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZETAN	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
EZETANH	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Note: *Includes Windows 95 and Windows 98, Windows NT, and Windows 2000.

Legend: In this table, the following characters are used to indicate the level of support:
x Supported
c Supported with compatibility considerations
blank Not supported

EZEABS

EZEABS returns the absolute value of *numericDataItem*.



Attribute	Description
result	Any numeric data item. The absolute value of *numericDataItem* is converted to the format of the result item and returned in the result.
numericDataItem	Any numeric data item.

Target environments for EZEABS

Supported in all environments without compatibility considerations.

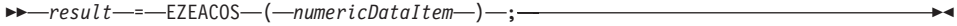
Example for EZEABS

```
EZEREPLY = 1;
RESULT = EZEABS(ITEM);
```

EZEACOS

EZEACOS is a function that returns the arccosine of *numericDataItem* in the range –1,1. The result, in radians, is in the range 0,*π*.

The function is implemented using the C function acos(*numericDataItem*).



Attribute	Description
result	Any numeric data item. The value returned by the acos function is converted to the format of the result item and returned in result.
numericDataItem	Any numeric data item. The data item is converted to a double precision floating point before the acos function is called.

Target environments for EZEACOS

Supported in all environments without compatibility considerations.

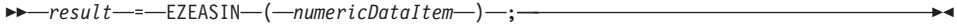
Example for EZEACOS

```
EZEREPLY = 0;
RESULT = EZEACOS(ITEM);
```

EZEASIN

EZEASIN is a function that returns the arcsine of *numericDataItem*, where *numericDataItem* is in the range –1,1. The result, in radians, is in the range $-\pi/2, \pi/2$.

This function is implemented using the C function asin(*numericDataItem*)



Attribute	Description
result	Any numeric data item. The value returned by the asin function is converted to the format of the result item and returned in result.

Attribute	Description
numericDataItem	Any numeric data item. The data item is converted to a double precision floating point before the asin function is called.

Target environments for EZEASIN

Supported in all environments without compatibility considerations.

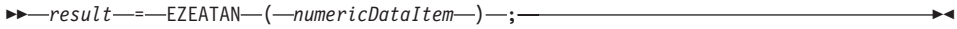
Example for EZEASIN

```
EZEREPLY = 1;
RESULT = EZEASIN(ITEM);
```

EZEATAN

EZEATAN is a function that returns the arctangent of *numericDataItem*. The result, in radians, is in the range $-\pi/2$, $\pi/2$.

This function is implemented using the C function atan(*numericDataItem*)



Attribute	Description
result	Any numeric data item. The value returned by the atan function is converted to the format of the result item and returned in result.
numericDataItem	Any numeric data item. The data item is converted to a double precision floating point before the atan function is called.

Target environments for EZEATAN

Supported in all environments without compatibility considerations.

Example for EZEATAN

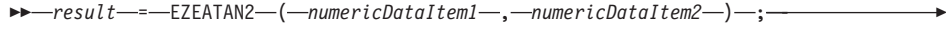
```
EZEREPLY = 0;
RESULT = EZEATAN(ITEM);
```

EZEATAN2

EZEATAN2 is a function that returns the theta component of the polar coordinate (r, theta) corresponding to the rectangular coordinate (*numericDataItem1*, *numericDataItem2*). The result, in radians, is in the range $-\pi$, π.

EZEATAN2

This function is implemented using the C function atan2(*numericDataItem1,numericDataItem1*).



Attribute	Description
result	Any numeric data item. The value returned by the atan2 function is converted to the format of the result item and returned in result.
numericDataItem1	Any numeric data item. The data item is converted to a double precision floating point before the atan2 function is called.
numericDataItem2	Any numeric data item. The data item is converted to a double precision floating point before the atan2 function is called.

Target environments for EZEATAN2

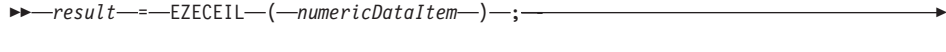
Supported in all environments without compatibility considerations.

Example for EZEATAN2

```
EZEREPLY = 1;
RESULT = EZEATAN2(ITEM1,ITEM2);
```

EZECEIL

EZECEIL is a function that returns the smallest integer not less than *numericDataItem*.



Attribute	Description
result	Any numeric data item. The absolute value of *numericDataItem* is converted to the format of the result item and returned in the result.
numericDataItem	Any numeric data item.

Target environments for EZECEIL

Supported in all environments without compatibility considerations.

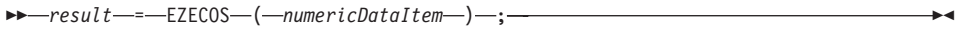
Example for EZECEIL

```
EZEREPLY = 0;
RESULT = EZECEIL(ITEM);
```

EZECOS

EZECOS is an operation that returns the cosine of *numericDataItem*. The result is in the range –1, 1.

The function is implemented using the C function cos(*numericDataItem*).



Attribute	Description
result	Any numeric data item. The value returned by the cos function is converted to the format of the result item and returned in result.
numericDataItem	Any numeric data item. The data item is converted to a double precision floating point before the cos function is called.

Target environments for EZECOS

Supported in all environments without compatibility considerations.

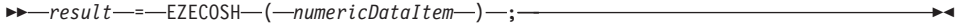
Example for EZECOS

```
EZEREPLY = 0;
RESULT = EZECOS(ITEM);
```

EZECOSH

EZECOSH is a function that returns the hyperbolic cosine of *numericDataItem*.

The function is implemented using the C function cosh(*numericDataItem*).



Attribute	Description
result	Any numeric data item. The value returned by the cosh function is converted to the format of the result item and returned in result.
numericDataItem	Any numeric data item. The data item is converted to a double precision floating point before the cosh function is called.

Target environments for EZECOSH

Supported in all environments without compatibility considerations.

EZECOSH

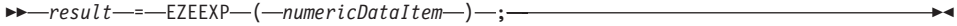
Example for EZECOSH

```
EZEREPLY = 0;
RESULT = EZECOSH(ITEM);
```

EZEEXP

EZEEXP is a function that returns the exponential value of *numericDataItem*. That is, *e* raised to the power of *numericDataItem*.

The function is implemented using the C function exp(*numericDataItem*).



Attribute	Description
result	Any numeric data item. The value returned by the exp function is converted to the format of the result item and returned in result.
numericDataItem	Any numeric data item. The data item is converted to a double precision floating point before the exp function is called.

Target environments for EZEEXP

Supported in all environments without compatibility considerations.

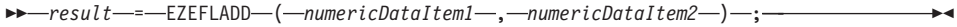
Example for EZEEXP

```
EZEREPLY = 0;
RESULT = EZEEXP(ITEM);
```

EZEFLADD

EZEFLADD is a function that returns the sum of *numericDataItem1* and *numericDataItem2*.

The function is implemented using double precision floating point arithmetic.



Attribute	Description
result	Any numeric data item. The sum is converted to the format of the result item and returned in result.

Attribute	Description
numericDataItem1	Any numeric data item. The numeric data item is converted to double precision floating point before the sum is calculated.
numericDataItem2	Any numeric data item. The numeric data item is converted to double precision floating point before the sum is calculated.

Target environments for EZEFLADD

Supported in all environments without compatibility considerations.

Example for EZEFLADD

```
EZEREPLY = 0;
RESULT = EZEFLADD(ITEM1,ITEM2);
```

EZEFLDIV

EZEFLDIV is a function that returns the quotient of *numericDataItem1* divided by *numericDataItem2*. A domain exception is raised if *numericDataItem2* is equal to 0.

The function is implemented using double precision floating point arithmetic.

Attribute	Description
result	Any numeric data item. The quotient is converted to the format of the result item and returned in result.
numericDataItem1	Any numeric data item. The numeric data item is converted to double precision floating point before the quotient is calculated.
numericDataItem2	Any numeric data item. The numeric data item is converted to double precision floating point before the quotient is calculated.

Target environments for EZEFLDIV

Supported in all environments without compatibility considerations.

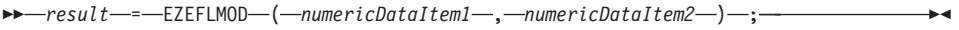
Example for EZEFLDIV

```
EZEREPLY = 0;
RESULT = EZEFLDIV(ITEM1,ITEM2);
```

EZEFLMOD

EZEFLMOD is a function that calculates the floating point remainder of *numericDataItem1* divided by *numericDataItem2*. The result has the same sign as *numericDataItem1*. A domain exception is raised if *numericDataItem2* is equal to 0.

The function is implemented using the C function fmod(*numericDataItem1*,*numericDataItem2*).



Attribute	Description
result	Any numeric data item. The floating point remainder is converted to the format of the result item and returned in result.
numericDataItem1	Any numeric data item. *numericDataItem1* is converted to double precision floating point before the fmod function is called.
numericDataItem2	Any numeric data item. *numericDataItem2* is converted to double precision floating point before the fmod function is called.

Target environments for EZEFLMOD

Supported in all environments without compatibility considerations.

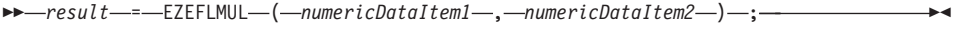
Example for EZEFLMOD

```
EZEREPLY = 0;
RESULT = EZEFLMOD(ITEM1,ITEM2);
```

EZEFLMUL

EZEFLMUL is a function that returns the product of *numericDataItem1* and *numericDataItem2*.

The function is implemented using double precision floating point arithmetic.



Attribute	Description
result	Any numeric data item. The product is converted to the format of the result item and returned in result.

Attribute	Description
numericDataItem1	Any numeric data item. The numeric data item is converted to double precision floating point before the product is calculated.
numericDataItem2	Any numeric data item. The numeric data item is converted to double precision floating point before the product is calculated.

Target environments for EZEFLMUL

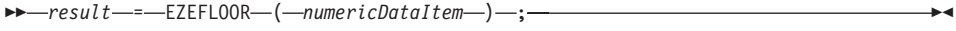
Supported in all environments without compatibility considerations.

Example for EZEFLMUL

```
EZEREPLY = 0;
RESULT = EZEFLMUL(ITEM1,ITEM2);
```

EZEFLOOR

EZEFLOOR is a function that returns the largest integer not greater than *numericDataItem*.



Attribute	Description
result	Any numeric data item. The largest integer not greater than *numericDataItem* is converted to the format of the result item and returned in result.
numericDataItem	Any numeric data item.

Target environments for EZEFLOOR

Supported in all environments without compatibility considerations.

Example for EZEFLOOR

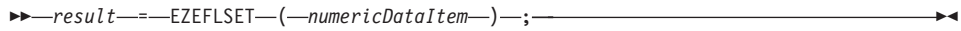
```
EZEREPLY = 0;
RESULT = EZEFLOOR(ITEM);
```

EZEFLSET

EZEFLSET is a function that converts *numericDataItem* to a double precision floating point number and then sets the result value from the floating point number.

The function assigns the value of BIN, NUM, NUMC, PACK, or PACKF items to floating point numbers defined as HEX items, and vice versa.

EZEFLSET



Attribute	Description
result	Any numeric data item. The floating point number is converted to the format of the result and returned in result.
numericDataItem	Any numeric data item. *numericDataItem* is converted to double precision floating point before it is assigned to the result.

Target environments for EZEFLSET

Supported in all environments without compatibility considerations.

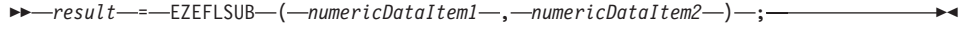
Example for EZEFLSET

```
EZEREPLY = 0;
RESULT = EZEFLSET(ITEM);
```

EZEFLSUB

EZEFLSUB is a function that returns the difference between *numericDataItem1* and *numericDataItem2*. *numericDataItem2* is subtracted from *numericDataItem1* and the difference is returned in result.

The function is implemented using double precision floating point arithmetic.



Attribute	Description
result	Any numeric data item. The difference is converted to the format of the result item and returned in result.
numericDataItem1	Any numeric data item. The numeric data item is converted to double precision floating point before the difference is calculated.
numericDataItem2	Any numeric data item. The numeric data item is converted to double precision floating point before the difference is calculated.

Target environments for EZEFLSUB

Supported in all environments without compatibility considerations.

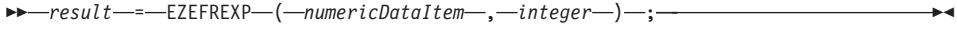
Example for EZEFLSUB

```
EZEREPLY = 0;
RESULT = EZEFLSUB(ITEM,INTEGER);
```

EZEFREXP

EZEFREXP is a function that splits *numericDataItem* into a normalized fraction in the range 1/2, 1, which is returned as the result, and a power of 2, which is stored in *integer*.

The function is implemented using the C function frexp(*numericDataItem*, *integer*).



Attribute	Description
result	Any numeric data item. The floating point fraction is converted to the format of the result item and returned in result.
numericDataItem	Any numeric data item. *numericDataItem* is converted to a double precision floating point number before frexp is called.
integer	An integer data item defined as 4-byte BIN with zero decimal places.

Target environments for EZEFREXP

Supported in all environments without compatibility considerations.

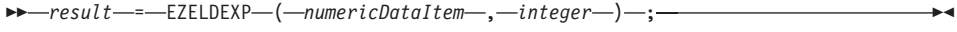
Example for EZEFREXP

```
EZEREPLY = 1;
RESULT = EZEFREXP(ITEM,INTEGER);
```

EZELDEXP

EZELDEXP is a function that returns the product of *numericDataItem* multiplied by 2 to the power of *integer*.

The function is implemented using the C function ldexp(*numericDataItem*, *integer*).



Attribute	Description
result	Any numeric data item. The floating point fraction is converted to the format of the result item and returned in result.

EZELDEXP

Attribute	Description
numericDataItem	Any numeric data item. *numericDataItem* is converted to a double precision floating point number before ldexp is called.
integer	An integer data item defined as 4-byte BIN with zero decimal places.

Target environments for EZELDEXP

Supported in all environments without compatibility considerations.

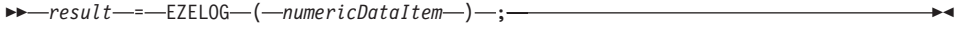
Example for EZELDEXP

```
EZEREPLY = 0;
RESULT = EZELDEXP(ITEM,INTEGER);
```

EZELOG

EZELOG is a function that returns the natural logarithm of *numericDataItem*.

The function is implemented using the C function log(*numericDataItem*).



Attribute	Description
result	Any numeric data item. The value returned by the log function is converted to the format of the result item and returned in result.
numericDataItem	Any numeric data item. The data item is converted to a double precision floating point before the log function is called.

Target environments for EZELOG

Supported in all environments without compatibility considerations.

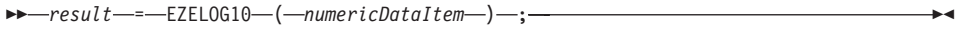
Example for EZELOG

```
EZEREPLY = 0;
RESULT = EZELOG(ITEM);
```

EZELOG10

EZELOG10 is a function that returns the base 10 logarithm of *numericDataItem*.

The function is implemented using the C function log10(*numericDataItem*).



Attribute	Description
result	Any numeric data item. The value returned by the log10 function is converted to the format of the result item and returned in result.
numericDataItem	Any numeric data item. The data item is converted to a double precision floating point before the log10 function is called.

Target environments for EZELOG10

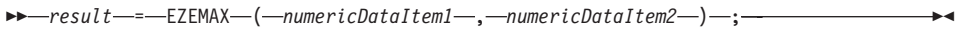
Supported in all environments without compatibility considerations.

Example for EZELOG10

```
EZEREPLY = 0;
RESULT = EZELOG10(ITEM);
```

EZEMAX

EZEMAX is a function that returns the maximum of *numericDataItem1* and *numericDataItem2*.



Attribute	Description
result	Any numeric data item. The maximum of *numericDataItem1* and *numericDataItem2* is converted to the format of the result item and returned in the result.
numericDataItem1	Any numeric data item.
numericDataItem2	Any numeric data item.

Target environments for EZEMAX

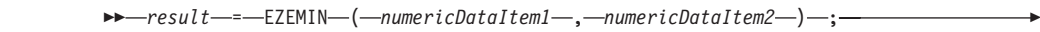
Supported in all environments without compatibility considerations.

Example for EZEMAX

```
EZEREPLY = 0;
RESULT = EZEMAX(ITEM1,ITEM2);
```

EZEMIN

EZEMIN is a function that returns the minimum of *numericDataItem1* and *numericDataItem2*.



Attribute	Description
result	Any numeric data item. The minimum of *numericDataItem1* and *numericDataItem2* is converted to the format of the result item and returned in the result.
numericDataItem1	Any numeric data item.
numericDataItem2	Any numeric data item.

Target environments for EZEMIN

Supported in all environments without compatibility considerations.

Example for EZEMIN

```
EZEREPLY = 0;
RESULT = EZEMIN(ITEM1,ITEM2);
```

EZEMODF

EZEMODF is a function that splits *numericDataItem1* into integral and fractional parts with the same sign as *numericDataItem1*.

The fractional part is returned in *result* and the integral part is returned in *numericDataItem2*.

Attribute	Description
result	Any numeric data item. The fractional part of *numericDataItem1* is converted to the format of the result item and returned in result.
numericDataItem1	Any numeric data item.

Attribute	Description
numericDataItem2	Any numeric data item. The integral part of *numericDataItem1* is converted to the format of *numericDataItem2* and returned in *numericDataItem2*.

Target environments for EZEMODF

Supported in all environments without compatibility considerations.

Example for EZEMODF

```
EZEREPLY = 0;
RESULT = EZEMODF(ITEM1,ITEM2);
```

EZENCMPR

EZENCMPR is a function that returns a result determined by comparing *numericDataItem1* to *numericDataItem2*.

Attribute	Description
result	An integer item defined as a 4-byte BIN with 0 decimal places. Result is set to the value of the result item and returned in result. The following values are returned: **1** numericDataItem1 is greater than numericDataItem2 **0** numericDataItem1 is equal than numericDataItem2 **-1** numericDataItem1 is less than numericDataItem2
numericDataItem1	Any numeric data item.
numericDataItem2	Any numeric data item.

Target environments for EZENCMPR

Supported in all environments without compatibility considerations.

Example for EZENCMPR

```
EZEREPLY = 0;
RESULT = EZENCMPR(ITEM1,ITEM2);
```

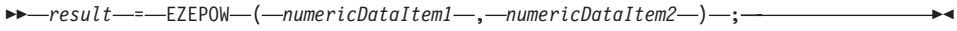
EZEPOW

EZEPOW is a function that returns *numericDataItem1* raised to the power of *numericDataItem2*.

EZEPOW

A domain exception is raised if *numericDataItem1* is equal to 0 and *numericDataItem2* is less than or equal to 0, or if *numericDataItem1* is less than 0 and *numericDataItem2* is not an integer.

The function is implemented using the C function pow(*numericDataItem1*,*numericDataItem2*).



Attribute	Description
result	Any numeric data item. The floating point fraction is converted to the format of the result item and returned in result.
numericDataItem1	Any numeric data item. *numericDataItem1* is converted to a double precision floating point number before the pow function is called.
numericDataItem2	Any numeric data item. *numericDataItem2* is converted to a double precision floating point number before the pow function is called.

Target environments for EZEPOW

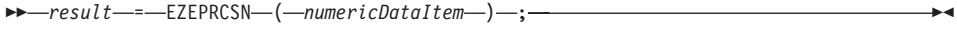
Supported in all environments without compatibility considerations.

Example for EZEPOW

```
EZEREPLY = 0;
RESULT = EZEPOW(ITEM1,ITEM2);
```

EZEPRCSN

EZEPRCSN is a function that returns the maximum precision in decimal digits for *numericDataItem*. For floating point numbers (4-byte HEX item for standard floating point number, 8-byte HEX for double precision floating point number), the precision is the maximum number of decimal digits that can be represented in the number for the system on which the program is running.



Attribute	Description
result	An integer item defined as a 4-byte BIN with 0 decimal places. On return result is set the precsion of *numericDataItem*.
numericDataItem	Any numeric data item.

Target environments for EZEPRCSN

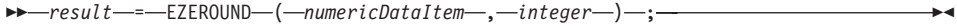
Supported in all environments without compatibility considerations.

Example for EZEPRCSN

```
EZEREPLY = 0;
RESULT = EZEPRCSN(ITEM);
```

EZEROUND

EZEROUND is a function that rounds *numericDataItem* to the *integer* power of 10 and returns the result.



Attribute	Description
result	Any numeric data item. The floating-point fraction is converted to the format of the result item and returned in result.
numericDataItem	Any numeric data item.
integer	An integer item defined as 4-byte BIN with 0 decimal places.

Target environments for EZEROUND

Supported in all environments without compatibility considerations.

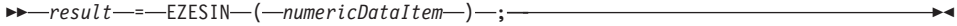
Example for EZEROUND

```
EZEREPLY = 0;
RESULT = EZEROUND(ITEM,INTEGER);
```

EZESIN

EZESIN is a function that returns the sine of *numericDataItem*. The result is in the range –1,1.

The function is implemented using the C function sin(*numericDataItem*).



Attribute	Description
result	Any numeric data item. The value returned by the sin function is converted to the format of the result item and returned in result.

EZESIN

Attribute	Description
numericDataItem	Any numeric data item. The data item is converted to a double precision floating point before the sin function is called.

Target environments for EZESIN

Supported in all environments without compatibility considerations.

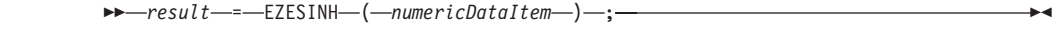
Example for EZESIN

```
EZEREPLY = 1;
RESULT = EZESIN(ITEM);
```

EZESINH

EZESINH is a function that returns the hyperbolic sine of *numericDataItem*.

The function is implemented using the C function sinh(*numericDataItem*).



Attribute	Description
result	Any numeric data item. The value returned by the sinh function is converted to the format of the result item and returned in result.
numericDataItem	Any numeric data item. The data item is converted to a double precision floating point before the sinh function is called.

Target environments for EZESINH

Supported in all environments without compatibility considerations.

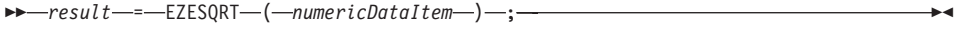
Example for EZESINH

```
EZEREPLY = 0;
RESULT = EZESINH(ITEM);
```

EZESQRT

EZESQRT is a function that returns the square root of *numericDataItem* where *numericDataItem* is greater than or equal to 0.

The function is implemented using the C function sqrt(*numericDataItem*).



Attribute	Description
result	Any numeric data item. The value returned by the sqrt function is converted to the format of the result item and returned in result.
numericDataItem	Any numeric data item. The data item is converted to a double precision floating point before the sqrt function is called.

Target environments for EZESQRT

Supported in all environments without compatibility considerations.

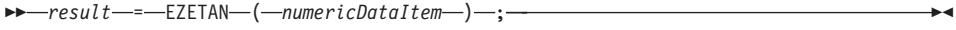
Example for EZESQRT

```
EZEREPLY = 0;
RESULT = EZESQRT(ITEM);
```

EZETAN

EZETAN is a function that returns the tangent of *numericDataItem*.

The function is implemented using the C function tan(*numericDataItem*).



Attribute	Description
result	Any numeric data item. The value returned by the tan function is converted to the format of the result item and returned in result.
numericDataItem	Any numeric data item. The data item is converted to a double precision floating point before the tan function is called.

Target environments for EZETAN

Supported in all environments without compatibility considerations.

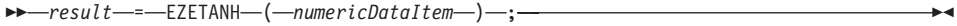
Example for EZETAN

```
EZEREPLY = 0;
RESULT = EZETAN(ITEM);
```

EZETANH

EZETANH is a function that returns the hyperbolic tangent of *numericDataItem*. The result is in the range –1,1.

The function is implemented using the C function tanh(*numericDataItem*).



Attribute	Description
result	Any numeric data item. The value returned by the tanh function is converted to the format of the result item and returned in result.
numericDataItem	Any numeric data item. The data item is converted to a double precision floating point before the tanh function is called.

Target environments for EZETANH

Supported in all environments without compatibility considerations.

Example for EZETAN

```
EZEREPLY = 0;
RESULT = EZETAN(ITEM);
```

Chapter 14. Object Scripting EZE words

The following Object Scripting EZE words are used to invoke a Smalltalk or Java script from within a VisualAge Generator function.

- EZESCRPT

Object scripting words

Table 26. Object Scripting words

Element	COBOL											GUI		C++								Test Facility
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	
EZESCRPT												x	x									c

Note: *Includes Windows 95 and Windows 98, Windows NT, and Windows 2000.

Legend: In this table, the following characters are used to indicate the level of support:
x Supported
c Supported with compatibility considerations
blank Not supported

EZESCRPT

EZESCRPT executes a script, written in the object-oriented language of your development environment, from within a VAGen function that was invoked by a GUI client. The scripts that can be invoked are associated with the instance of the GUI client class and cannot require arguments. The invocation is executed synchronously, like a CALL statement or any other VisualAge Generator function invocation.

Uses

Attribute	Description
target	The target can be either the name of the method being invoked from the VAGen function or the name of a data item that contains the string literal name of the method. The target is case sensitive.

Some example uses of EZESCRPT are:

- To access user interface parts on a GUI client
- To access Enterprise JavaBeans (VisualAge for Java)

Definition considerations

Scripts can be invoked only from functions that are executed from the GUI client with which the VAGen Script is stored. They cannot be invoked from server programs.

EZESCRPT accepts one argument: a literal script name or the name of the data item that contains the script name. Script names are case sensitive. To preserve the case of a literal script name you specify, enclose it in double quotes as shown in the example.

Script names should adhere to the following rules:

- Script names can contain numbers, but the first character must be a letter.
- Spaces and special characters are not valid in script names.
- In Smalltalk, DBCS, and mixed script names are not valid.
- In Java, character, DBCS, and mixed script names are all valid.

For information on packaging GUI client programs and details about object scripting requirements, refer to the VisualAge Generator User's Guide.

Target environments for EZESCRPT

Environment	Compatibility considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	Not supported.
MVS/TSO	Not supported.
MVS batch	Not supported.
IMS/VS	Not supported.
IMS BMP	Not supported.
CICS for VSE/ESA	Not supported.
VSE batch	Not supported.

Environment	Compatibility considerations
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2 (GUI)	None.
Windows (GUI)	None.
OS/2 (C++)	Not supported.
AIX	Not supported.
HP-UX	Not supported.
CICS for AIX	Not supported.
Windows NT	Not supported.
CICS for Windows NT	Not supported.
Test Facility	None. See the online help for details on how Test Facility handles script invocations.

Example for EZESCRPT

```
IF DATA_ITEM1 NOT NUMERIC;
   EZESCRPT("indicateErrorInText");
END;
```

```
IF DATA_ITEM1 NOT NUMERIC;
   EZESCRPT(dataItem);
END;
```

Chapter 15. User interface EZE words

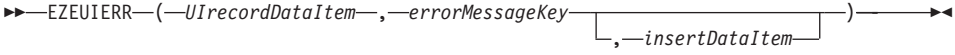
The following User Interface (UI) EZE words are available for use in Web Transaction programs:

- EZEUIERR
- EZEUILOC

EZEUIERR

EZEUIERR is used to set a given field in a UI record in error. Use of EZEUIERR is restricted to Web Transactions.

If EZEUIERR is invoked in a user defined edit, the UI record will be CONVERSEd again. Invocation of the method hasInputError() for the UI Record Bean as a whole or on the specific item that is set in error returns true.



Attribute	Description
DataItem	The UI record data item that is to be set in error.
ErrorMessageKey	A CHA or MIX item or literal that contains the error message key to associate with the error. This is the key into a user message table that contains an error message. If this key is blank, there is no lookup done and the message is just a concatenation of the inserts.
insertDataItem	One or more data items that contain the inserts for the referenced error message. The insertDataItem is inserted in the message text where **%n** is found for the **nth** insertDataItem.

Target environments for EZEUIERR

EZEUIERR is available for Web Transactions only.

Example for EZEUIERR

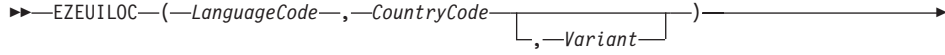
```
EZEUIERR ( UIRECORD.INPUT_FIELD, "MSGE1001", WSRECORD.INSERT_VALUE ) ;
```

EZEUILOC

EZEUILOC is used to change UI Bean Locale.

This function is used to programmatically change the Locale being used by the program. Each invocation would change the Locale for the duration of the user session. Edit routines can be run to invoke EZEUILOC and change the Locale for the duration of the Web Transaction Program. The Locale change occurs at the web server where the NLS data of the UI Record Bean is accessed by JavaServer Pages. This function is only allowed to run in an edit function that runs at the web server in a UI bean.

EZEUILOC conforms to the JDK 1.1 API documentation for class java.util.Locale. See ISO 639 for language codes and ISO 3166 for country codes.



Attribute	Description
LanguageCode	CHA data item or literal with two character language code. Only language codes defined by ISO 639 are valid.
CountryCode	CHA data item or literal with two character country code. Only country codes defined by ISO 3166 are valid.
Variant	CHA data item or literal with indefinite length variant. Variant code selection depends on browser and vendor products. Specifying the variant is optional.

Target environments for EZEUILOC

EZEUILOC is available for Web Transactions only.

Example for EZEUILOC

```
EZEUILOC("fr","CA") ;
```

Chapter 16. Services

VisualAge Generator services are system-dependent services that VisualAge Generator programs can call in various operating environments. If portability between supported and unsupported environments is required, you can develop a non-VisualAge Generator program with the same name as the service routine to receive the service call in the unsupported environments. The program either does nothing and returns to the caller, or it simulates the service function in the unsupported environment.

Services elements

Table 27. Services elements

Element	COBOL											GUI			C++									Java	
	VM CMS	VM Batch	MVS CICS	MVS/TSO	MVS Batch	IMS/VS	IMS BMP	VSE CICS	VSE Batch	OS/2 CICS	OS/400	OS/2	Java	Windows*	OS/2	AIX	HP-UX	AIX CICS	Windows NT	Windows NT CICS	Solaris	Solaris CICS	Windows NT	Test Facility
AUDIT			c		c	c	c	c																x
COMMIT (same as EZECOMIT)	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	x
CREATX			c			c	c	c		c	c	c			c	c	c	c	c	c	c	c	c	x
CSPTDLI			c	x	x	c	x	c	c															c
EZCHART	x		x	x				x																x
RESET (same as EZEROLLB)	c	c	c	c	c	c	c	c	c	c	x	c	c	c	c	c	c	c	c	c	c	c	c	c

Note: *Includes Windows 95 and Windows 98, Windows NT, and Windows 2000.

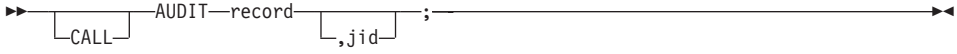
Legend: In this table, the following characters are used to indicate the level of support:
x Supported
c Supported with compatibility considerations
blank Not supported

AUDIT

AUDIT writes a record to the CICS journal or IMS log.

Uses

The program can use AUDIT to write tracking information to the system log or journal.



Attribute	Description
record	The name of a record to be written to a journal file. The first 2 bytes contain the length of the record to be written. The next 2 bytes contain a user-supplied code identifying the source of the journal record. The first byte of the user code must be in the range X'A0' to X'FF'. In addition to containing the record length and record identifier code, the first 28 bytes are reserved for system usage. They should not contain user data because the data is overlaid when it is written to the terminal. Bytes 29 to 32767 are available for user audit data.
jid	An optional parameter that specifies the ID (1-99) of the journal file to which the service routine writes the record. If jid is omitted, the record is written, by default, to the system journal. The parameter is a 2-byte binary number.

Target environments for AUDIT

Environment	Compatibility Considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	If the REPLY option is specified on the CALL, EZERT8 contains one of the following codes after the CALL: **Code** **Description** **000** Successful completion **201** Length error **202** User source code error **204** Journal identifier error **803** I/O Error
MVS/TSO	Not supported.

Environment	Compatibility Considerations
MVS batch	Not supported unless a PSB is specified for the program and the program does at least one of the following: • Uses EZEDLPSB or EZEDLPCB in any statement in the program • Has DL/I databases other than ELAWORK or ELAMSG in the PSB definition • Uses CSPTDLI • Associates at least one file or EZEPRINT with GSAM. If a PSB is specified and the program does one of the above, same as IMS/VS.
IMS/VS	The record is automatically converted to IMS log format, by VisualAge Generator by adding 2 to the length and inserting 2 bytes of binary zeros following the length field. Only the first byte of the record identifier code is used. The second byte of the record identifier code is ignored. The jid parameter is ignored. IMS/VS has a maximum limit of 32765 bytes.
IMS BMP	Same as IMS/VS.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
Java (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
CICS for AIX	Not supported.
HP-UX	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.

Environment	Compatibility Considerations
Test Facility	None.

Examples for AUDIT

In this example, the data structure, WRKSTG, is shown in the following table:

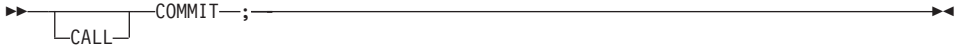
Name	Level	Occurs	Type	Length	Bytes	
WHOLE	05	1	CHA	32765	32765	/*WHOLE RECORD
HEADER	10	1	CHA	28	28	/*HEADER
LENGTH	15	1	BIN	4	2	/*RECORD LENGTH (BINARY)
CODE	15	1	CHA	2	2	/*2 CHARACTER CODE
RSRVD1	15	1	CHA	24	24	/*RESERVED
DATA	10	1	CHA	32737	32737	/*USER SUPPLIED DATA
JRNLID	77	1	BIN	4	2	/*JOURNAL ID (BINARY LEVEL-77)

The following code shows the data structure written to journal file number 2:

```
MOVE 32765 TO LENGTH;
MOVE 'A' TO CODE;
MOVE 2 TO JRNLID;
MOVE 'THIS IS THE DATA TO BE WRITTEN TO JOURNAL NUMBER 2' TO DATA;
AUDIT WRKSTG,JRNLID;
```

COMMIT

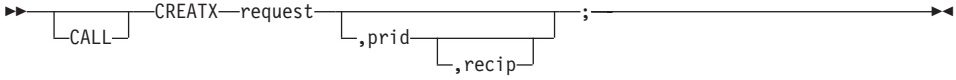
The COMMIT service is equivalent to the EZECOMIT special function word. However, the syntax for using COMMIT is different than the syntax for using EZECOMIT.



Refer to EZECOMIT for further information.

CREATX

CREATX starts an asynchronous transaction in a transactional environment and passes a record to the transaction being started. If the transaction is a VisualAge Generator program, the record is used to initialize the working storage of the program.



Attribute	Description
request	The name of a working-storage record. The working-storage record must have the following format: • The first 2 bytes (binary) contain the length of the data to be passed to the started transaction, plus 10 for the length and transaction name fields. These 2 bytes are not passed to the started transaction. The maximum length cannot exceed 32767 bytes. • The next 8 bytes (character) contain the name of the transaction to be started. These 8 bytes are not passed to the started transaction. **Note:** The transaction name is system-dependent and is described in the compatibility considerations. If the transaction invokes a VisualAge Generator program, the program must be type main. • The remaining part of the request record is passed to the started transaction.
prid	An optional 4-character or 4-byte binary item. Its use is system dependent and is described in the Target environments section. If you specify `recip`, `prid` is required.
recip	An optional 4-character item containing the terminal ID to be associated with the transaction being started.

Definition considerations for CREATX

You can use the CREATX service in CICS environments to start a transaction on a remote system and pass a record to the transaction for processing. The remote program receives the record in the working storage area when the transaction is started.

At program generation, use the linkage table to specify that the CREATX statement starts a transaction at a remote system. The linkage table also specifies how the location of the remote system is to be determined and whether the data in the working storage record needs to be converted between mainframe and workstation format. Refer to the section on implementing cooperative processing using the CREATX service routine in the *VisualAge Generator Design Guide* document for more information.

The prid and recip arguments are ignored if specified on a CREATX call to a remote program. The started transaction is not associated with any terminal.

Target environments for CREATX

Environment	Compatibility Considerations
VM CMS	Not supported.
VM batch	Not supported.
CICS for MVS/ESA	The 4-byte CICS transaction code should be followed by 4 bytes of blanks in the transaction name field in the request record. The 2-byte length and the transaction name field are not passed to the started transaction. The CREATX results in a CICS START command being issued for the indicated transaction. The START command is issued with or without an associated terminal, depending on the contents of `recip`. If `recip` is omitted, the started transaction is associated with the current terminal. The value of `prid` is ignored. If PRINTDEST=TERMID and `recip` is set to binary zeros, the value of `prid` becomes the initial value for EZEDESTP in the started transaction. The CICS START command is issued without an associated terminal. If PRINTDEST=TERMID, EZEDESTP will be initialized to the following: • If main batch program, EZEDESTP initialized to EIBTRMID • If main transaction, EZEDESTP initialized to destination associated with EZEPRINT at generation. • If called program, the same guidelines are followed unless the caller also displays printer maps, in which case the called program always initializes with same destination as the calling program. If PRINTDEST=EZEP, EZEDESTP will always be initialized to the destination associated with EZEPRINT at generation. If `recip` is not binary zeros, it must contain the terminal ID to be associated with the transaction being started. It can be either a terminal or a printer ID. It is not recommended that this be the current terminal. Use XFER to start a new transaction on the current terminal. The value of `prid` is ignored. The started transaction must have PCT and PPT entries for the program. If the started transaction is not a VisualAge Generator program, it must issue a CICS RETRIEVE to get the passed work area.

Environment	Compatibility Considerations
CICS for MVS/ESA (continued)	If the REPLY option is specified on the CALL, EZERT8 can contain the following return codes: **Code** **Description** **00000000** Successful CREATX **00000203** Transaction identifier not valid **00000205** Terminal identifier not valid **00000206** Parameters not valid **00000207** System identifier not valid **00000208** Link out of service **ffrrrrrr** Other CICS error where **ff** is the hexadecimal representation of EIBFN byte 0 and **rrrrrr** is the hexadecimal representation of EIBRCODE bytes 0-2.
MVS/TSO	Not supported.
MVS batch	Not supported.

Environment	Compatibility Considerations
IMS/VS	The transaction name field in the request record is the 8-byte IMS transaction code. CREATX results in an insert to the modifiable alternate PCB. The indicated work area is passed as the message. The generated COBOL program automatically adds an extra 2 bytes between the length and the transaction and adds 2 to the length value. The transaction is started without an associated terminal. `Prid` and `recip` are ignored. The maximum length on the request record is 32765 bytes. The transaction that is started must be included in the IMS GEN and must be defined as a nonconversational transaction. If the started transaction is not a VisualAge Generator program, it must issue a get unique call to the I/O PCB to retrieve the message. CREATX is not supported to a program at a remote system or from a program at a remote system. If the REPLY option is specified on the call, EZERT8 can contain the following value: **Code Description** **00000203** CREATX failed. The most probable cause is a transaction identifier that is not valid.
IMS BMP	Same as IMS/VS.
CICS for VSE/ESA	Same as CICS for MVS/ESA.
VSE batch	Not supported.

Environment	Compatibility Considerations
CICS for OS/2	Be sure to follow the 4-byte CICS transaction code by 4 bytes of blanks in the transaction name field in the request record. The CREATX results in a CICS START command being issued for the indicated transaction. The START command is issued with or without an associated terminal, depending on the contents of `recip`. If `recip` is omitted, the started transaction is associated with the current terminal. The value of `prid` is ignored. If `recip` is not binary zeros, it must contain the terminal ID to be associated with the transaction being started. It can be either a terminal or a printer ID. It is not recommended that this be the current terminal. Use XFER to start a new transaction on the current terminal. The value of `prid` is ignored. The started transaction must have a PCT entry for the program. If the started transaction is not a VisualAge Generator program, it must issue a CICS RETRIEVE to get the passed work area. The `prid` argument is ignored if specified. The CREATX results in a CICS START command being issued for the indicated transaction. The transaction is started without an associated terminal. Refer to CICS for MVS/ESA for a list of return codes in EZERT8 if the REPLY option is specified on the CALL.

Environment	Compatibility Considerations
OS/400	VisualAge Generator Server for AS/400 provides OS/400 support for CREATX by way of command language (CL) programs (PGMs). Two CL PGMs in the VisualAge Generator Server for AS/400 product support the CREATX function, CREATX and CREATXPP. CREATX gets the current job number and sends the user data to a data queue called VGCREATX. CREATX then starts a new job called CREATXJOB, which starts the CREATXPP CL PGM. CREATXPP uses the previous job number as the key to retrieve the data from the data queue VGCREATX. CREATXPP then calls the asynchronous CL PGM specified in the user data record bytes 3 through 11. Define the CREATX call arguments as follows: **VG RECORD** The maximum record length is 4095 bytes for OS/400 CREATX calls. Bytes 1 through 2 is the length of the VisualGen data being passed, plus 10 for the length field and the program name field. Bytes 3 through 11 is the name of the asynchronous program. Bytes 12 through 4095 is the actual VisualGen user data. **PRID** A 4-byte char field with the value of the output queue used for the asynchronous job. The default value is VGEN. (This output queue must be defined before executing the first CREATX call.) **RECIP** This parameter is not used in AS/400 CREATX, but should be specified for environment compatibility.
OS/2 (GUI)	The `prid` and `recip` arguments are ignored if specified.
Windows (GUI)	Not supported.
Java (GUI)	Not supported.
OS/2 (C++)	Same as OS/2 (GUI).
AIX	Same as OS/2 (GUI).

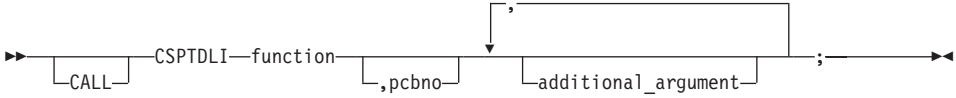
Environment	Compatibility Considerations
CICS for AIX	Be sure to follow the 4-byte CICS transaction code by 4 bytes of blanks in the transaction name field in the request record. The CREATX results in a CICS START command being issued for the indicated transaction. The START command is issued with or without an associated terminal, depending on the contents of `recip`. If `recip` is omitted, the started transaction is associated with the current terminal. The value of `prid` is ignored. If `recip` is not binary zeros, it must contain the terminal ID to be associated with the transaction being started. It can be either a terminal or a printer ID. It is not recommended that this be the current terminal. Use XFER to start a new transaction on the current terminal. The value of `prid` is ignored. The started transaction must have a PCT entry for the program. If the started transaction is not a VisualAge Generator program, it must issue a CICS RETRIEVE to get the passed work area. The `prid` argument is ignored if specified. The CREATX results in a CICS START command being issued for the indicated transaction. The transaction is started without an associated terminal. Refer to CICS for MVS/ESA for a list of return codes in EZERT8 if the REPLY option is specified on the CALL.
HP-UX	Same as OS/2 (GUI).
Solaris	Same as OS/2 (GUI).

Environment	Compatibility Considerations
CICS for Solaris	Be sure to follow the 4-byte CICS transaction code by 4 bytes of blanks in the transaction name field in the request record. The CREATX results in a CICS START command being issued for the indicated transaction. The START command is issued with or without an associated terminal, depending on the contents of `recip`. If `recip` is omitted, the started transaction is associated with the current terminal. The value of `prid` is ignored. If `recip` is not binary zeros, it must contain the terminal ID to be associated with the transaction being started. It can be either a terminal or a printer ID. It is not recommended that this be the current terminal. Use XFER to start a new transaction on the current terminal. The value of `prid` is ignored. The started transaction must have a PCT entry for the program. If the started transaction is not a VisualAge Generator program, it must issue a CICS RETRIEVE to get the passed work area. The `prid` argument is ignored if specified. The CREATX results in a CICS START command being issued for the indicated transaction. The transaction is started without an associated terminal. Refer to CICS for MVS/ESA for a list of return codes in EZERT8 if the REPLY option is specified on the CALL.
Windows NT (C++)	Same as OS/2 (GUI).

Environment	Compatibility Considerations
Windows NT (Java)	The prid and recip arguments are ignored if specified. CREATX may only be used for UI records and local Java Server Programs. Java programs can only use CREATX to start other VAGen Java programs. The program created using CREATX will run in a separate JVM, using the command specified in the property vgj.java.command. The default command is java. The package of a program created using CREATX comes from the package attribute in the linkage table used at generation. If it is not supplied, it defaults to the package of the calling program. If the REPLY option is specified on the CALL, and a new JVM cannot be created or an error occurs while passing the record to the new program, EZERT8 will contain 00000001. If the new JVM can be created and the record is successfully passed to the new program, EZERT8 will contain 00000000. Note that it is possible for EZERT8 to contain 00000000 even if the new program fails to run.
CICS for Windows NT	Same as CICS for AIX.
Test Facility	None.

CSPTDLI

CSPTDLI enables you to issue any DL/I call that is supported by the execution environment.



Attribute	Description
function	The name of a 4-character variable or a literal containing a DL/I function code.
pcbno	The name of a 2-byte binary variable containing the number of the PCB to be used on the DL/I or IMS/DC call. The I/O PCB is PCB 0. All other PCBs are numbered according to the order in which they appear in the PSB.

Attribute	Description
additional_argument	A DL/I call parameter. The additional_argument parameter can be a character, mixed literal or DBCS literal, record, map, or data item. The definition and contents of the additional_argument must match the definition of the parameters that are required for the function code. Certain CSPTDLI calls may not require additional arguments. If a CSPTDLI call requires additional arguments, the first additional_argument is always treated as the IO area. You can specify a maximum of 18 arguments on the call. Up to 15 additional_arguments are allowed following the function, pcbno and IO area arguments.

Definition considerations for CSPTDLI

You can use the CSPTDLI service to execute any DL/I function, including those supported through I/O options or special function words. In a specific environment, only those DL/I functions that are supported in the environment execute correctly. DL/I returns a nonblank status code in the PCB for calls that are not supported in the execution environment.

To check the status information returned by DL/I or IMS/DC after the call, the program should move the PCB to a working storage area using the EZEDLPCB special function word.

A PSB is required for the program to use CSPTDLI.

VisualAge Generator does not validate the DL/I call. You must code a valid DL/I call.

Target environments for CSPTDLI

Environment	Compatibility Considerations
VM CMS	Not supported.
VM batch	Not supported.

Environment	Compatibility Considerations
CICS for MVS/ESA	Do not use CSPTDLI to schedule (PCB function code) or release (TERM function code) a PSB. VisualAge Generator automatically handles PSB scheduling. Only DB PCBs can be used with CSPTDLI in the CICS environment. Specify the PCB number based on the order of the PCB in the PSB part definition. DL/I call parameters must be 24-bit addressable if IMS/VS is the product that supports DL/I calls. The IMS/ESA installation option must be set to **No** if you use IMS/VS with VisualAge Generator Server for MVS, VSE, and VM. Refer to the *VisualAge Generator Server Guide for MVS, VSE, and VM* document for information on setting the IMS/ESA installation option.
MVS/TSO	None.
MVS batch	None.
IMS/VS	Do not use the following in a transaction program or in a batch program that is called by a transaction program: • Get unique or insert call to the I/O PCB • ROLB call • CHKP call
IMS BMP	None.
CICS for VSE/ESA	Do not use CSPTDLI to schedule (PCB function code) or release (TERM function code) a PSB. The generated program automatically handles PSB scheduling. Specify the PCB number based on the order of the PCB in the PSB part definition. Do not adjust the number for any TP PCBs in the PSB part definition that are not included in the DL/I DOS/VS PSB. Only function codes that are supported by DL/I DOS/VS execute correctly. Refer to the DL/I DOS/VS program programming manual for a description of the supported codes.
VSE batch	Only function codes that are supported by DL/I DOS/VS execute correctly. Refer to the DL/I DOS/VS program programming manual for a description of the supported codes. Specify the PCB number based on the order of the PCB in the PSB part definition. Do not adjust the number for any TP PCBs in the PSB part definition that are not included in the DL/I DOS/VS PSB.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.

Environment	Compatibility Considerations
Java (GUI)	Not supported.
OS/2 (C++)	Not supported.
AIX	Not supported.
CICS for AIX	Not supported.
HP-UX	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Test Facility	Calls to I/O or TP PCBs using CSPTDLI cannot be tested or emulated on the workstation. DL/I calls that access IMS message queues or fast path databases cannot be tested under the test facility. You must set up the local DL/I database or remote DL/I communications link before testing DL/I access. Refer to the *VisualAge Generator User's Guide* for more information.

Examples for CSPTDLI

Example 1

The example uses the restart (XRST) IMS DL/I function.

Note: You can restore up to 7 data areas. This example only uses one. The example also assumes that you would specify the checkpoint ID in the PARM field of the program's JCL, not by coding it in the program.

To issue a restart (XRST) call in a batch-oriented BMP:

1. Define the following data items:

Table 28. Data items for CSPTDLI example

Name	Type	Length	Description
IOPCBNUM	BIN	2	Number of the I/O PCB
IOASIZE	BIN	4	Size of the I/O area (record in message queue)
CHKPID	CHA	8	Checkpoint ID

Table 28. Data items for CSPTDLI example (continued)

Name	Type	Length	Description
AREALEN1	BIN	4	Length of first data area
DATA	CHA	50	First data area to restore

2. Type the following sample code in your program:

```
/* initialize values */
   MOVE 0 TO IOPCBNUM;
   MOVE 12 TO IOASIZE;
   MOVE 50 TO AREALEN1;
   MOVE " " TO CHKPID;

   CSPTDLI "XRST",IOPCBNUM,IOASIZE,CHKPID,AREALEN1,DATA;
   MOVE EZEDLPCB[0] TO IOPCB;   /* Using item in DLIIOPCB record */
                                /* that VisualAge Generator      */
                                /* ships, then check status      */
                                /* codes and checkpoint ID       */
```

Example 2

The example uses CSPTDLI to perform the equivalent of an INQUIRY function, retrieving the record where employee number equals 20.

Note: You should use the INQUIRY function with a modified segment search argument when you want to implement the functionality of the example in other circumstances.

To issue a get unique (GU) call:

1. Define the following data items:

Table 29. Data items for CSPTDLI example

Name	Type	Length	Description
FUNC_CODE	CHA	4	Function code
PCB_NO	BIN	2	Number of the I/O PCB
RD03NMED	RECORD	100	I/O area
SSA_1	CHA	26	SSA string

2. Type the following sample code in your program:

```
/* initialize values */
MOVE 'GU  ' TO FUNC_CODE;
MOVE 3 TO PCB_NO;
MOVE 'RD03NMED(EMPNO   EQ000020)' TO SSA_1;
```

```
/* actual CSPTDLI call - will retrieve the data with employee */
/* number equal to 20 and place it in the ioarea, RD03NMED. */
CALL CSPTDLI FUNC_CODE, PCB_NO, RD03NMED, SSA_1;
```

EZCHART

EZCHART enables you to access the Interactive Chart Utility (ICU) of the Graphic Data Display Manager (GDDM) from a VisualAge Generator program.

Uses

A call to EZCHART is essentially a call to a non-VisualAge Generator program that converts the X and Y axis data to floating point and passes it on to the ICU. Some parameter checking is done in EZCHART and return codes are set accordingly. The ICU does not return any information to the calling program. All facilities normally available in the ICU are available to the VisualAge Generator program user.

Definition considerations for EZCHART

Some knowledge of the ICU is useful when using EZCHART. Refer to the *Presentation Graphics Feature (PGF) Reference* for more information on the ICU. The introduction explains many of the terms used, and the CALL description gives an explanation of the parameters passed by VisualAge Generator Server for MVS, VSE, and VM.

A VisualAge Generator program that uses EZCHART can be one that prompts the program user for data through several maps, then puts that data into the necessary format and shows the result to the program user in chart form. From there, the program user might be allowed to alter the chart, depending upon the DISPLAY option that was indicated in the parameter list by the VisualAge Generator program. Programs that allow an interactive session with the ICU, assume that the program user is familiar the ICU, because VisualAge Generator has no control over the ICU session.

A VisualAge Generator program can be programmed to collect and rearrange user data and then print the results in chart form, possibly with several types of charts incorporating the same data. In which case, the use of EZCHART may be entirely transparent to the program user, because PRINT might be an option from a VisualAge Generator map that causes a CALL to EZCHART with the PRINT option and then returns to the VisualAge Generator program.

Analysts can use the VisualAge Generator programs to enter the ICU with collected data and modify the charts. These charts can then be used in other VisualAge Generator programs.

The option to display a chart with no ICU session can be useful for demonstrations and meetings, especially if the output can be projected on a large screen.

Parameters for EZCHART

You can optionally specify the CALL statement before EZCHART. The following parameters must be specified after EZCHART in the following order:

1. Return code
2. Chart control
3. Data control
4. X-axis data
5. Y-axis data
6. Keys
7. Labels
8. Heading

Return code, chart control, and data control are required parameters. Other parameters are optional.

You can leave out unused parameters from the end of the list when calling EZCHART and the dummy data is passed to ICU, but parameters cannot be left out of the middle of the list. For example, when labels are to be used, XDATA must still be passed even though the information in this record will not be used.

Parameter List Validation

EZCHART takes the passed parameter list and does the following:

- Checks the number of actual parameters passed against the information in Chart Control. For example, if the KEYL field in Chart Control is not zero, all parameters up through the KEYS parameter must be present.
- The X-axis and Y-axis data are converted to floating point data. However, if the LABEL field is not zero, the X-axis data is ignored, because the label data is used to build the chart.

Return code parameter for EZCHART

Return code is a 4-byte numeric field for completion code. This field cannot be defined as binary, and should be checked for a non-zero value upon return from the call to EZCHART.

The following error return codes can be received from EZCHART:

2 One or more of the 3 required parameters was not passed, or:
KEYL > 0, but no KEYS record was passed, or:
LABELL > 0, but no LABELS record was passed, or:
HEADINGL > 0, but no HEADING record was passed, or:
PAIRING > 0, but no XDATA was passed.

3 Unable to get storage for X or Y parameters

4 Overflow on floating point conversion

8 Missing printer copies

9 Missing printer name

10 Load or link to GDDM failed

Chart control parameter for EZCHART

Chart control is a structure of specified format containing control information necessary to run the ICU session:

LEVEL
Binary item with length of 9 (BIN 4 bytes) which must be 0

DISPLAY
BIN 4 bytes; a number from 0 to 7 controlling the ICU session

The DISPLAY portion of the Chart Control parameter controls the type of ICU session that is invoked. There are eight DISPLAY options offered by the ICU; All are supported transparent to VisualAge Generator:

DISPLAY=0
This option builds a Chart Data File using the data supplied in the other parameters. No ICU session is started and no chart is built.

DISPLAY=1
This option enters the ICU at the home panel, having initialized the chart and data formats using the parameters passed.

DISPLAY=2
This option displays the requested chart with the data passed from the program, and then allows the program user to use the ICU facilities interactively to modify and save the chart.

DISPLAY=3
This option displays the requested chart with the data passed from the program, but limits the program user's access to the ICU menus.

DISPLAY=4
This is the print option that prints the specified map on the specified device and returns control to the program.

DISPLAY=5
This option uses the parameters to construct a chart, but no device I/O is performed.

DISPLAY=6
This option is the same as option 5, except the user can modify the chart.

DISPLAY=7
This option builds a chart from the parameters passed and sends it to the current output device. No ICU menu panels are used.

HELP BIN 4 bytes; possible values are 0 and 1, where 1=help information is on the panel when it first appears

ISOLATE
BIN 4 bytes; possible values are 0 and 1, where 0=user can SAVE and RESTORE in ICU session

FORMNAME
8-character name of a previously defined chart format, or an asterisk to indicate a line graph

DATANAME
8-character name of previously saved data or asterisk to indicate that data is being passed on the CALL

If DATANAME is equal to an asterisk, you must pass Y-axis data on a call to EZCHART. Otherwise, unpredictable results can occur.

PAIRING
BIN 4 bytes; possible values are 0 and 1, where 1=passed data that is paired

NG BIN 4 bytes; number of groups in the data being passed

NE BIN 4 bytes; maximum number of elements in any group

KEYL BIN 4 bytes; length of keys passed

LABELL
BIN 4 bytes; length of labels passed as X-axis data

HEADINGL
BIN 4 bytes; length of character string heading

PRTNAME
8-character name of local print destination

PRTDEP
BIN 4 bytes; depth of chart on printer

PRTWID
BIN 4 bytes; width of chart on printer

PRTCOPY
BIN 4 bytes; number of copies to be printed.

Data control parameter for EZCHART

Data control is an array of items that indicate the number of elements in each chart group. This must be a separate record definition, defined as BIN 4 bytes, with the OCCURS value set to the number of groups (NG) as specified in the control chart. The ICU facility requires data that is passed to be in short floating point form. You need not be familiar with this form, but you should be aware that some loss of precision might occur in the conversion of fractional parts of numbers, due to truncation.

X-axis data parameter for EZCHART

A record definition, with:

- An item defined as BIN 4 bytes containing the length of the field.
- An item defined as BIN 4 bytes containing the number of decimal places.
- A NUM item in which the X-axis data is passed. The length in bytes is the value of the first item in the record. The number of occurrences is the sum of all elements for all the groups.

Y-axis data parameter for EZCHART

A record definition, with:

- An item defined as BIN 4 bytes containing the length of the field.
- An item defined as BIN 4 bytes containing the number of decimal places.
- A NUM item in which the Y-axis data is passed. The length in bytes is the value of the first item in the record. The number of occurrences is the sum of all elements for all the groups.

Missing data values in the X-axis and Y-axis: The ICU provides a means for indicating missing data values in the X-axis and Y-axis arrays. If you want to specify missing data values in your EZCHART call, you must move blanks to any array elements for which there is no data available. To do this, you must substructure your array elements as character fields as shown in the following example:

```
Item Name   Level  Occurs  Type  Length   Dec    Bytes
  YLENG      10      1      BIN      9      0       4
  YDEC       10      1      BIN      9      0       4
  YITEMS     10      xx     NUM     yy     zz      yy
  Y-MISSING  15      1      CHA     yy     zz      yy
```

In the example, xx is the number of items specified in the Data Control record, yy is the value in YLENG, and zz is the value in YDEC. Any missing values can be set by the following statement:

```
MOVE ' ' to Y-MISSING[subscript];
```

The EZCHART routine passes these values to the ICU as 10 raised to the power of 72. The ICU recognizes this value as an indicator of a missing data value. Refer to the *Presentation Graphics Feature (PGF) Reference* to determine the effects on different types of charts.

When using paired data, do not specify missing values in the X-axis array. A GDDM error will occur.

Keys parameter for EZCHART

Is a record containing an array of keys for data groups to be used in building a chart. There must be as many keys as there are groups in the chart. This item must be in a separate record and must be defined as CHA with key length specified in Chart Control and OCCURS as the number of groups specified in Chart Control.

Labels parameter for EZCHART

Is a record containing an array of character strings that relate to labels along the X-axis of certain types of charts (bar, pie, etc.) that are used instead of X-axis data in building the chart. This item must be in a separate record and must be defined as CHA with label length specified in Chart Control and OCCURS as the number of elements specified in Chart Control.

Heading parameter for EZCHART

Is a record containing a character string that is used to build a heading on the requested chart. This must be a record defined as CHA (heading length specified in Chart Control).

Target environments for EZCHART

Environment	Compatibility Considerations
VM CMS	None.
VM batch	Not supported.
CICS for MVS/ESA	None.
MVS/TSO	None.
MVS batch	Not supported.
IMS/VS	Not supported.
IMS BMP	Not supported.
CICS for VSE/ESA	None.
VSE batch	Not supported.
CICS for OS/2	Not supported.
OS/400	Not supported.
OS/2 (GUI)	Not supported.
Windows (GUI)	Not supported.
Java (GUI)	Not supported.
OS/2 (C++)	Not supported.

Environment	Compatibility Considerations
AIX	Not supported.
CICS for AIX	Not supported.
HP-UX	Not supported.
Solaris	Not supported.
CICS for Solaris	Not supported.
Windows NT (C++)	Not supported.
Windows NT (Java)	Not supported.
CICS for Windows NT	Not supported.
Test Facility	None.

Examples for EZCHART

To issue a call to EZCHART, you must first define the following records:

1. WS_RECD - Working storage record for EZCHART example

```
Item Name  Level Occurs Type Bytes      Description
Return_code  77    1    NUM   4   /* 0=CALL was successful        */
```

2. CHART_CTRL_RECD - Working storage record for EZCHART example

```
Item Name  Level Occurs Type Bytes      Description
  LEVEL     10     1    BIN   4   /* Must be zero                 */
  DISPLAY   10     1    BIN   4   /* ICU access control number    */
  HELP      10     1    BIN   4   /* 0=PFkey info not displayed   */
  ISLATE    10     1    BIN   4   /* 0=User can save and restore  */
  FORM      10     1    CHA  8   /* Name of saved chart or '*'   */
  DATA      10     1    CHA  8   /* Name of saved data or '*'    */
  PAIRING   10     1    BIN   4   /* 1=Paired data (x and y)      */
  NGRPS     10     1    BIN   4   /* Number of data groups        */
  NELEMS    10     1    BIN   4   /* Number of data elements (max)*/
  KEYL      10     1    BIN   4   /* Length of keys passed        */
  LABELL    10     1    BIN   4   /* Length of labels passed      */
  HDINGL    10     1    BIN   4   /* Length of heading            */
  PRTNM     10     1    CHA  8   /* Local print destination name */
  PRTDEP    10     1    BIN   4   /* Depth of chart on printer    */
  PRTWID    10     1    BIN   4   /* Width of chart on printer    */
  PRCOPY    10     1    BIN   4   /* Number of copies to print    */
```

3. DATA_CTRL_RECD - Working storage record for EZCHART example

```
Item Name  Level Occurs Type Bytes      Description
  DCNTL      10    ?     BIN   4   /* Number of elements in each   */
                                   /* group.                       */
                                   /* Occurs =     value of        */
                                   /*        CHART_CTRL_RECD.NGRPS */
```

4. X_DATA_RECD - Working storage record for EZCHART example

```
Item Name  Level Occurs Type Bytes Dec      Description
  XLENG     10     1    BIN   4    0 /* Total length of field       */
  XDEC      10     1    BIN   4    0 /* Number of decimal places    */
  XITEMS    10     ?    NUM   ?    ? /* Array of X-items,           */
                                     /* Occurs= total of all DCNTL  */
                                     /*         ... + DCNTL[n]      */
                                     /*         items in DATA_CT */
                                     /*         ... RL_RECD      */
                                     /* Bytes = value of XLENG      */
                                     /* Dec   = value of XDEC       */
```

5. Y_DATA_RECD - Working storage record for EZCHART example

```
Item Name  Level Occurs Type Bytes Dec      Description
  YLENG    10      1    BIN   4    0 /* Total length of field       */
  YDEC     10      1    BIN   4    0 /* Number of decimal places    */
  YITEMS   10      ?    NUM   ?    ? /* Array of Y-items,           */
                                     /* Occurs= total of all DCNTL  */
                                     /*         ... + DCNTL[n]      */
                                     /*         items in DATA_CT */
                                     /*         ... RL_RECD      */
                                     /* Bytes = value of YLENG      */
                                     /* Dec   = value of YDEC       */
```

6. KEY_ARRAY - Working storage record for EZCHART example

```
Item Name  Level Occurs Type Bytes      Description
  KEYARR    10     ?    CHA  ?   /* Occurs = value of           */
                                  /*    CHART_CTRL_RECD.NGRPS     */
                                  /* Bytes = value of            */
                                  /*    CHART_CTRL_RECD.KEYL      */
```

7. LABEL_ARRAY - Working storage record for EZCHART example

```
Item Name  Level Occurs Type Bytes      Description
  LABLAR    10     ?    CHA  ?   /* Occurs = value of           */
                                  /*    CHART_CTRL_RECD.NELEMS    */
                                  /* Bytes = value of            */
                                  /*    CHART_CTRL_RECD.LABELL    */
```

8. HEAD_RECD - Working storage record for EZCHART example

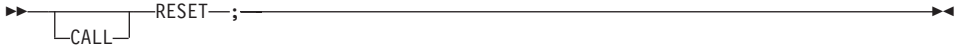
```
Item Name  Level Occurs Type Bytes      Description
  HEAD      10     1    CHA  ?   /* Bytes = value of            */
                                  /*    CHART_CTRL_RECD.HDINGL    */
```

Initialize the data values in the records used as parameters. Then call EZCHART using the following:

```
CALL EZCHART  RETURN_CODE, CHART_CTRL_RECD, DATA_CTRL_RECD,
              X_DATA_RECD, Y_DATA_RECD, KEY_ARRAY, LABEL_ARRAY,
              HEAD_RECD;
```

RESET

The RESET service is equivalent to the EZEROLLB special function word. However, the syntax for using RESET is different than the syntax for using EZEROLLB.



Refer to EZEROLLB for further information.

Part 3. Appendixes

Appendix A. Reading syntax diagrams

The syntax diagrams used throughout the documentation conform to the following conventions:

- The keywords can be listed in any order.

  The ►►— symbol indicates the beginning of a statement.

  The —► symbol indicates that the statement syntax is continued on the next line.

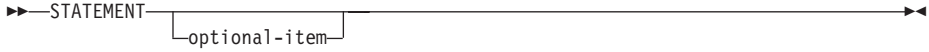
  The ►— symbol indicates that a statement is continued from the previous line.

  The —►◄ symbol indicates the end of a statement.

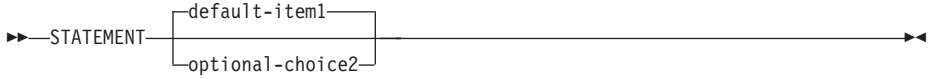
  A syntax diagram that does not show the complete statement starts with the ►— symbol and ends with the —► symbol. The >>— symbol indicates the beginning of a statement.

  A syntax diagram that show the complete statement starts with the ►►— symbol and ends with the —►◄ symbol.
- Required items appear on the horizontal line (the main path).

- Optional items appear below the main path.



- Items positioned above the syntax diagram line are default parameters.



- If you can choose from two or more items, these items appear vertically, in a stack.

  If you *must* choose an item in the stack, one of the required items appears on the main path.

If choosing one of the items is optional, the entire stack appears below the main path.

- An arrow returning to the left above the item indicates an item that you can repeat. Required items appear on the main line and optional items appear below the main line.

  A repeat arrow indicates that you can make more than one choice from the grouped items, or repeat a single item.
- Keywords appear in uppercase (for example, `PARM1`). However, they can be uppercase or lowercase when they are entered. They must be spelled exactly as shown. Variables and acceptable values appear in all lowercase letters (for example, `parmx`). They represent names or values that you supply.
- If punctuation marks, parentheses, arithmetic operators, or other symbols are shown, you must enter them as part of the syntax.

Appendix B. Naming conventions for data item, record, function names

The following are the naming conventions for data item, record, and function names:

Maximum length
18 (for record and function)

Maximum length
32 (for data item)

First character
alphabetic (A-Z) or one of the valid national characters for your workstation

See National characters for a description of national characters in part names.

Other characters
alphanumeric (A-Z, 0-9), underscore (_), hyphen (-), or one of the valid national characters for your workstation

DBCS name
Yes, maximum length: 8 (for record and function)

DBCS name
Yes, maximum length: 15 (for data item)

See DBCS naming conventions for a description of DBCS part names.

- An asterisk can be used as the data item name in any data structure. This type of data item is called a filler data item, which cannot be referenced by the program. It acts as a space holder in a data structure.
- A data item name can be used only once within a single data structure.

The following conventions apply to all part names:
- Part names cannot begin with the EZE prefix.
- Part names cannot contain embedded blanks.

To avoid aliases being assigned during COBOL generation and to improve the readability of the generated COBOL program, use a name that meets the following COBOL naming conventions:
- Use 30 characters or less in data item names.
- Do not use COBOL reserved words.
- Do not use @, #, $, or _ characters.
- Do not use DBCS names for record or function names.

- Do not use DBCS names for data item names if your program contains SQL functions.

Data items in DL/I records: Data item names used in DL/I records are limited to 8 characters. They cannot be DBCS names or contain hyphens or underscores. To use long names for data items in a DL/I record, define a redefined record for the DL/I segment and use long names for the fields in the redefined record. Use the DL/I segment definition only as the I/O object and in defining DL/I calls.

National characters

By default, the English language version of VisualAge Generator recognizes the following three code points as valid national characters in part names:

Code point
Symbol

Hex 24
Dollar sign ($)

Hex 23
Number or pound sign (#)

Hex 40
At sign (@)

The set of national characters you can use might differ from those in the list above, depending on the following:

- Whether you are using a language version other than English
- Whether you have customized the national characters in the EZERDEV.NLS file for your workstation

Note: Avoid using these characters if the program you are developing will be exported or generated for another code page.

Refer to the *VisualAge Generator Installation Guide* document for more information about valid national characters.

DBCS naming conventions

VisualAge Generator supports DBCS names for any part with a name that can be longer than 8 characters. A valid DBCS name must meet the following conditions:

- The DBCS part name can contain only DBCS characters.

- The DBCS part name cannot contain SBCS characters.
- The DBCS part name cannot contain a DBCS blank.
- The maximum number of DBCS characters in a name is shown in the following table:

PART NAME	MAXIMUM
Function	8
Record	8
Data item	15

- A DBCS name must contain at least one DBCS character that does not have a SBCS equivalent (non-42nd-ward DBCS character). The only valid SBCS-equivalent (42nd-ward) DBCS characters are as follows:
  - double-byte A through Z (.A — .Z)
  - double-byte 0 through 9 (.0 — .9)
  - double-byte **@**, **#**, **$**, _ and **-** (hyphen)

  Double-byte lowercase characters a-z are folded to double-byte uppercase A-Z when used in a DBCS name.

Note: A 42nd-ward DBCS character contains Hexadecimal 42 in the first byte when translated to EBCDIC.

The following table shows DBCS names that are valid and not valid:

VALID DBCS NAMES	NOT VALID DBCS NAMES
.CDi.B	.A.B.C
DiDjDk	AB.C

- Do not use a DBCS name for a function or record.
- Do not use a DBCS name for a data item name if your program contains SQL functions.

Appendix C. Size restrictions and record lengths

Size limitations for VisualAge Generator

Table 30 outlines size limitations for VisualAge Generator. Refer to specific language element compatibility considerations for additional environmental restrictions.

Table 30. Size Limitations for VisualAge Generator

Definition	Limitations
Number of Data items	32767 data items and literals per program
Data items	32767 bytes in record definition 32730 bytes in record definition (OS/400 only) 254 bytes in table definition 8000 bytes for printer maps (IMS only) 1 byte less than map size for terminal maps (IMS only) 990 data items per record
Map Constant Field	255 bytes (IMS only)
Working Storage	32767 bytes if used in an XFER or DXFR statement 32730 bytes maximum, regardless of XFER or DXFR (OS/400 only)
Numeric Items	18 digits
Decimal Places	18 digits (included within numeric item size)
Subscripting	One level
Number of Occurrences	32767 in record definition 32730 in record definition (OS/400 only)
Maximum Table	524288 bytes for MVS, VM, VSE, or non-shared tables on CICS for OS/2 64K bytes for shared tables on CICS for OS/2 On OS/400, table rows are limited to 32,767 bytes, total table contents is limited to 3 mega bytes.
Maximum Number of Variable fields on a Map	800 on CICS for OS/2
Primary Table Columns	700 top level data items

Table 30. Size Limitations for VisualAge Generator (continued)

Definition	Limitations
Numeric Literals	18 digits plus 1 sign, 1 decimal point, or both
Comments	60 characters of comment per prologue line 73 characters of comment per line or statement
CALL Parameters	Limit of 30 arguments
Number of Main Functions	254 per program
Number of lines in an SQL statement	819

Maximum record lengths

Table 31 outlines the maximum record lengths for each environment.

Table 31. Maximum record lengths by environment when using an XFER statement

Environment	XFER with Record	XFER with Record and Map
VM CMS	32767	32767
VM batch	32767	XFER not supported
CICS (main or main batch)	32763 (limit set by CICS)	32753 (Main only) - (32763 - 10 bytes reserved for VisualAge Generator Server for MVS, VSE, and VM)
MVS/TSO (main or main batch)	32767	32767 (Main only)
MVS batch (main batch)	32767	XFER not supported
IMS/VS (main)	32753	32753
IMS/VS (main batch)	XFER not supported	XFER not supported
IMS BMP (main batch)	32767	XFER not supported
OS/400 (main or main batch)	32730	32730
VSE batch	32767	XFER not supported
OS/2	32767	XFER not supported
Windows NT	32767	XFER not supported
AIX	32767	XFER not supported
HP-UX	32767	XFER not supported
Solaris	32767	XFER not supported

Table 32. Maximum record lengths by environment when using a DXFR statement

Environment	DXFR with Record
VM CMS	32767
VM batch	32767
CICS	32763
MVS/TSO	32767
MVS batch	32767
IMS/VS	32767
IMS BMP	32767
VSE	32767
OS/400	32730
VSE batch	32767
OS/2	32767
Windows NT	32767
AIX	32767
HP-UX	32767
Solaris	32767

Table 33. Maximum record lengths for serial, relative, indexed, message queue and working storage records by environment

Environment	Serial, Relative, Indexed, Message Queue and Working Storage Records
CICS (VSAM)	32763 (32688 for journaled records) 9999 for serial and indexed records on CICS for OS/2 4092 for relative records on CICS for OS/2
CICS (TD queue)	32763 (32767 on CICS for OS/2)
CICS (TS queue)	32762
CICS (Spool)	32763
IMS/VS (message queue)	32755 (32767 - 2 bytes for LL, 2 bytes for ZZ, and 8 bytes for transaction name) (IMS/VS Main can only ADD to a message queue)

Table 33. Maximum record lengths for serial, relative, indexed, message queue and working storage records by environment (continued)

Environment	Serial, Relative, Indexed, Message Queue and Working Storage Records
IMS BMP (main batch)	32755 (32767 - 2 bytes for LL, 2 bytes for ZZ, and 8 bytes for transaction name)
OS/400	32730

Table 34. The maximum audit data length for a record by environment

Environment	Audit Data Length
CICS	32763
MVS batch (with DL/I)	32765 (32767 - 2 bytes ZZ)
IMS/VS (with DL/I)	32765 (32767 - 2 bytes ZZ)
IMS BMP (with DL/I)	32765 (32767 - 2 bytes ZZ)

Table 35. The maximum CREATX data length for a record by environment

Environment	CREATX Data Length
CICS	32763
IMS/VS	32765 (32767 - 2 bytes ZZ)
IMS BMP	32765 (32767 - 2 bytes ZZ)

Index

F

H

I

R

S

T

U

V

W

X

Z

Readers' Comments — We'd Like to Hear from You

VisualAge Generator
Programmer's Reference
Version 4.5

Publication No. SH23-0262-01

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
SH23-0262-01

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

SH23-0262-01

Cut or Fold
Along Line

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SH23-0262-01