

VisualAge Generator



Migration Guide

Version 4.0

Note

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

First Edition (October 1999)

This edition applies to the following licensed programs:

- IBM VisualAge Generator Developer for OS/2 and Windows NT Version 4.0
- IBM VisualAge Generator Server for OS/2, AIX, Windows NT, HP-UX, and Solaris Version 4.0
- IBM VisualGen Host Services for OS/400 Version 3.1
- IBM VisualGen Host Services for OS/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) 284-4721.

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/vagen>

By FAX, use the following number:

- United States and Canada: 919-254-0206
- Other countries: 1-919-254-0206

By mail to the following address:

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

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

© **Copyright International Business Machines Corporation 1997, 1999. All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	ix
Trademarks	xi
About This Document	xiii
Who Should Use This Document	xiii
How to Use This Document	xiii
Terminology used in this document	xiii
Terminology differences between Java and Smalltalk	xiv
Documentation provided with VisualAge Generator	xv

Part 1. Migrating from VAGen 3.x to VAGen 4.0 on Java 1

Chapter 1. General migration considerations for VAGen 3.x to Java	3
Migration paths	3
Overview of the V3 to V4 Migration Tool on Java	4
Automatic conversions during 4.0 migration	4
Migrating GUIs	5
Migrating applications, subapplications, and configuration maps	6
Migrating VAGen Templates	7
Establishing naming conventions	8
Assigning ownership	10
Storing control information	10
Generation options	11
Dual maintenance	12
Migrating from OS/2 to Windows NT	12
Chapter 2. Pre-migration checklist	15
Chapter 3. Using the V3 to V4 Migration Tool to migrate VAGen 3.x code and VAGen Templates parts to Java	17
Setting V3 to V4 Migration Tool options on Java	18
Selection Criteria options	19
Java Naming Convention option	21
Smalltalk Subapplications options	22
Library Management options	23

Ownership options	24
Selecting and migrating applications and configuration maps	24
Working with the status log	26
Resetting the V3 to V4 Migration window	28
Resetting Migration Status Information	29

Chapter 4. Using VAGen Import to migrate VAGen 3.x non-GUI code to Java 31

Chapter 5. Completing the ENVY setup on Java 33

Chapter 6. Completing your migration on Java	35
Defining control information	35
Generating programs	35
Importing work-in-progress	35
Recreating ITF resource association information	36
Converting an RTABLE to a Linkage Table	36

Part 2. Migrating from VAGen 2.x or Cross System Product to VAGen 4.0 on Java 37

Chapter 7. Comparing MSLs and library management on VAGen 4.0 on Java	39
ENVY characteristics	39
Comparison of MSLs and ENVY	41
Member types	41
Storing members	42
Storing control information	42
MSL concatenation	43
Functional organization	44
Member associations	44

Chapter 8. General migration considerations for VAGen 2.x and Cross System Product to Java	47
Migration paths	47
Automatic conversions during 4.0 migration	48
Migrating GUIs	49
Resolving duplicate member names	49

Establishing naming conventions	50	Multiple subsystems with controlled duplicates.	89
Organizing your code for ENVY	52	Recommendations	89
Functional organization	52	Separate production MSLs for each developer	92
Assigning ownership	53	Recommendations	93
Storing control information	54	MSLs that contain unintended duplicates	97
Multiple control packages	54	Recommendations	98
Single control package	55	MSLs containing code from VisualAge Generator Templates or BW*Wizard	98
Generation options.	56	Recommendations	100
Dual maintenance	57	Special considerations for VisualAge Generator Templates.	102
Migrating from Cross System Product	57	Complete set of MSLs for production and deltas for test	102
Migrating from OS/2 to Windows NT	58	Recommendations	104
Using the migration log file.	58	Complete sets of MSLs for test and production	106
Chapter 9. Pre-migration checklist.	59	Recommendations	108
Chapter 10. The MSL Migration Assistance Tool on Java	61	MSLs from marketing or other demonstrations.	111
Overview of using the MSL Migration Assistance Tool	62	Recommendations.	112
Building MSL directories.	63	Chapter 13. Running the MSL Migration Assistance Tool on Java	115
Selecting MSLs - using the MSL Library Selection window	64	Starting VisualAge Generator	115
Selecting parts for a part list - using the Part List Selection Criteria View window	65	Creating users and setting the current user	116
Selecting parts to move to ENVY - using the MSL Migration Part List window	67	Collecting your source code	117
Working in the sandbox - using the VG Part Prerequisites View window	70	From Cross System Product	117
Reloading previously migrated ENVY packages back into the MSL Migration Assistance Tool sandbox	72	From VisualAge Generator with TeamConnection and no MSLs	117
Chapter 11. Migrating production and work-in-progress MSLs to VAGen 4.0 on Java	75	From VisualAge Generator MSLs.	117
Production MSLs	76	Handling code page changes	118
Techniques for moving parts to the sandbox	77	Using the HPTRULES.NLS file	118
Work-in-progress MSLs	80	Changing from OS/2 to Windows NT	119
Using VAGen Import	81	Starting the MSL Migration Assistance Tool	121
Using the MSL Migration Assistance Tool	82	Building MSL directories	121
Chapter 12. VAGen on Java case studies based on various MSL structures	85	Resetting the sandbox from ENVY	123
Understanding the diagrams and terminology	85	Selecting your MSLs	125
MSL structure diagrams	85	Selecting and migrating VAGen parts	126
Advance command in VisualAge Generator	86	Creating a new package.	129
Multiple subsystems with no duplicates.	87	Moving a VAGen part between packages	129
Recommendations	87	Controlling the creation of package.nodes	130
		Renaming a package.	131
		Collapsing a package	132
		Handling Duplicates.	132
		Controlled Duplicates	133
		Unintended Duplicates	133
		Duplicates for business logic	136
		Finding the package in which a part is located	137

Listing missing (not found) parts	138
Handling missing (not found) parts	139
Checking relationships among packages	140
Determining which programs are referenced	140
Determining the parts that are referenced	141
Checking consistency of packages	141
Updating the list of required packages	142
Changing the list of required packages	142
Normalizing the list of required packages	143
Deleting a package.node	143
Deleting a package	144
Deleting one package	144
Deleting all packages	145
Committing to ENVY	145

Chapter 14. Using VAGen Import to migrate VAGen 2.x and Cross System Product non-GUI code to Java. 149

Chapter 15. Completing the ENVY setup on Java. 151

Versioning and releasing a VAGen part class	151
Versioning and releasing a package	152
Versioning a project	153
Creating a Project List Part (PLP)	154
Changing the owner of a project	155
Assigning ownership of a VAGen part class	155
Adding group members	155
Changing the ownership of a VAGen part class	156
Changing the owner of a package	156

Chapter 16. Completing your migration on Java 159

Defining control information	159
Generating Programs	160
Importing work-in-progress	161
Migrating VSAM files	163
Converting an RTABLE to a Linkage Table	163

Part 3. Migrating from VAGen 3.x to VAGen 4.0 on Smalltalk. 165

Chapter 17. General migration considerations for VAGen 3.x to 4.0 on Smalltalk 167

Migration paths	167
---------------------------	-----

Overview of the V3 to V4 Migration Tool on Smalltalk.	168
Automatic conversions during 4.0 migration	169
Migrating GUIs	170
Migrating applications, subapplications, and configuration maps	170
Migrating VAGen Templates	171
Establishing naming conventions.	172
Assigning ownership	173
Using subapplications	173
Storing control information	173
Dual maintenance.	173
Migrating from OS/2 to Windows NT	174

Chapter 18. Pre-migration checklist 175

Chapter 19. Using the V3 to V4 Migration Tool to migrate VAGen 3.x code and VAGen Templates parts on Smalltalk 177

Setting V3 to V4 Migration Tool options on Smalltalk.	178
Selection Criteria options	179
Library Management options	181
Ownership options	182
Selecting and migrating applications and configuration maps	183
Working with the status log	185
Resetting the V3 to V4 Migration window	187
Resetting Migration Status Information	188

Chapter 20. Using VAGen Import to migrate VAGen 3.x non-GUI code to Smalltalk 189

Chapter 21. Completing the ENVY setup on Smalltalk 191

Versioning and releasing a view or a VAGen part class	191
Versioning an application	192
Creating a configuration map	193
Adding a required map to a configuration map	194
Versioning a configuration map	195
Changing the manager of a configuration map	195
Testing a configuration map	196
Assigning ownership of a VAGen part class	196
Adding group members	196
Changing the ownership of a VAGen part class	197

Changing the manager of an application . . . 197

Chapter 22. Completing your migration on Smalltalk 199
Defining control information 199
Generating programs 199
Importing work-in-progress 199
Recreating ITF resource association information 200
Converting an RTABLE to a Linkage Table 200

Part 4. Migrating from VAGen 2.x or Cross System Product to VAGen 4.0 on Smalltalk 203

Chapter 23. Comparing MSLs and ENVY on VAGen 4.0 on Smalltalk 207
ENVY characteristics 207
Comparison of MSLs and ENVY 209
 Member types 209
 Storing members 210
 Storing control information 210
 MSL concatenation 211
 Functional organization 212
 Member associations 212

Chapter 24. General migration considerations for VAGen 2.x and Cross System Product to Smalltalk 215
Migration paths 215
Automatic conversions during 4.0 migration 216
Resolving duplicate member names 217
Establishing naming conventions 217
Organizing your code for ENVY 219
Assigning ownership 220
Using subapplications 221
Storing control information 223
 Multiple control applications 223
 Single control application 224
 Generation options 225
Using configuration maps 225
Migrating GUIs 226
 Conversion of GUIs to VisualAge Generator 4.0 228
Dual maintenance 235
Migrating from Cross System Product 236
Migrating from OS/2 to Windows NT 237
Using the migration log file 237

Chapter 25. Pre-migration checklist 239

Chapter 26. The MSL Migration Assistance Tool on Smalltalk 241
Overview of using the MSL Migration Assistance Tool 242
 Building MSL directories 243
 Selecting MSLs - using the MSL Library Selection window 244
 Selecting parts for a part list - using the Part List Selection Criteria View window 245
 Selecting parts to move to ENVY - using the MSL Migration Part List window 246
 Working in the sandbox - using the VG Part Prerequisites View window 250
Resetting the MSL Migration Assistance Tool sandbox 252

Chapter 27. Migrating production and work-in-progress MSLs to VAGen 4.0 on Smalltalk 255
Production MSLs 256
 Techniques for moving parts to the sandbox 257
Work-in-progress MSLs 261
 Using VAGen Import 263
 Using the MSL Migration Assistance Tool 264

Chapter 28. VAGen on Smalltalk case studies based on various MSL structures . 265
Understanding the diagrams and terminology 265
 MSL structure diagrams 265
 Advance command in VisualAge Generator 266
Multiple subsystems with no duplicates 267
 Recommendations 267
Multiple subsystems with controlled duplicates 269
 Recommendations 269
Separate production MSLs for each developer 272
 Recommendations 273
MSLs that contain unintended duplicates 277
 Recommendations 278
MSLs containing code from VisualAge Generator Templates or BW*Wizard 278
 Recommendations 280
 Special considerations for VisualAge Generator Templates 282

Complete set of MSLs for production and deltas for test	282
Recommendations	284
Complete sets of MSLs for test and production	285
Recommendations	287
MSLs from marketing or other demonstrations	289
Recommendations	290

Chapter 29. Running the MSL Migration Assistance Tool on Smalltalk 293

Starting VisualAge Generator	293
Creating users and setting the current user	294
Loading a feature	295
Collecting your source code	296
From Cross System Product	296
From VisualAge Generator with TeamConnection and no MSLs	296
From VisualAge Generator MSLs	297
Handling code page changes	297
Using the HPTRULES.NLS file	297
Changing from OS/2 to Windows NT	298
Starting the MSL Migration Assistance Tool	300
Building MSL directories	300
Resetting the sandbox from ENVY	302
Selecting your MSLs	304
Selecting and migrating VAGen parts	305
Creating a new application.	308
Moving a VAGen part between applications	308
Controlling the creation of ApplicationNodes	310
Renaming an application	311
Collapsing an application	311
Handling Duplicates	312
Controlled Duplicates	312
Unintended Duplicates	313
Duplicates for business logic	315
Finding the application in which a part is located	316
Listing missing (not found) parts	317
Handling missing (not found) parts.	318
Checking relationships among applications	319
Determining which programs are referenced	319
Determining the parts that are referenced	320
Checking consistency of applications	321
Updating the list of required applications	321
Changing the list of required applications	322
Normalizing the list of required applications.	322

Deleting an ApplicationNode	323
Deleting an application	324
Deleting one application	324
Deleting all applications	324
Committing to ENVY	325

Chapter 30. Using VAGen Import to migrate VAGen 2.x and Cross System Product code to Smalltalk 329

Chapter 31. Completing the ENVY setup on Smalltalk 331

Versioning and releasing a view or a VAGen part class	331
Versioning an application	332
Creating a configuration map	333
Adding a required map to a configuration map	334
Versioning a configuration map	335
Changing the manager of a configuration map	335
Testing a configuration map	335
Assigning ownership of a VAGen part class	336
Adding group members	336
Changing the ownership of a VAGen part class	336
Changing the manager of an application	337

Chapter 32. Completing your migration on Smalltalk 339

Defining control information	339
Generating programs and packaging views	340
Importing work-in-progress	341
Migrating VSAM files	343
Converting an RTABLE to a Linkage Table	344

Chapter 33. Hints and tips on Smalltalk 345

System Transcript Window.	345
VisualAge Organizer window.	345
VAGen Parts Browser window	345
Refreshing the MSL Migration Assistance Tool	346

Part 5. Sharing VAGen 4.0 parts between Java and Smalltalk . . . 349

Chapter 34. Sharing VAGen 4.0 parts between Java and Smalltalk 351

Moving 4GL parts from Java to Smalltalk	351
---	-----

Moving 4GL parts from Smalltalk to Java	352	Collecting Information before Migration	392
Chapter 35. Sharing VAGen Templates 4.0 specifications between Java and Smalltalk	355	Contact List	393
Moving VisualAge Generator Templates specifications from Java to Smalltalk	355	Appendix D. Notes on Cross System Product migrations.	395
Moving VisualAge Generator Templates specifications from Smalltalk to Java	357	Running Cross System Product applications with VisualAge Generator Server for workstation platforms	395
Part 6. Appendixes	359	Updated Chapter 8, CSP/370RS 1.1 to VisualAge Generator Server for MVS, VSE, and VM	396
Appendix A. Name changes for parts and part classes	361	Installation considerations	396
Appendix B. Planning for Migrating Cross System Product MSLs to ENVY	365	Procedures	396
Collecting Information about Your Environment	365	Upward compatibility — generating applications, tables, and map groups again	397
Collecting Information before Migration	378	Error routines	397
Contact List	379	Defining PSBs	397
Appendix C. Planning for Migrating VAGen MSLs to ENVY.	381	User messages	397
Collecting Information about Your Environment	381	Glossary	399
		Index.	411

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:

AD/Cycle
AIX
AS/400
C Set ++
C/370
CICS
CICS/ESA
CICS for MVS/ESA
CICS for OS/2
CICS for VSE/ESA
CICS for AIX
COBOL/370
COBOL/400
DB2
DB2/VSE
DB2/2
DB2/400
DB2/6000
IBM
IMS
IMS/ESA
Language Environment
MVS
MVS/ESA
Operating System/2
OS/2
OS/400
PROFS
RACF
SAA
SQL/DS
SQL/400
TeamConnection
Virtual Machine/Enterprise Systems Architecture
VisualGen
VisualAge
VM
VM/ESA
VTAM

The following terms are trademarks or products of other companies:

Acrobat	Adobe Systems Incorporated
Adobe	Adobe Systems Incorporated
C++	American Telephone & Telegraph Company
ENVY	Object Technology International Inc.
HP-UX	Hewlett-Packard Company
Lotus Notes	Lotus Development Corporation
Solaris	Sun Microsystems, Inc.

BW*Wizard is a product of Bridgewater Consultants, Inc. It is protected under US copyright laws and can be used only under a license from Bridgewater Consultants, Inc.

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

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

About This Document

This document provides information needed to migrate to VisualAge Generator 4.0 on Smalltalk and VisualAge Generator 4.0 on Java from VisualAge Generator 3.x, VisualAge Generator 2.x, and Cross System Product.

Who Should Use This Document

This document assumes knowledge either of Cross System Product or of VisualAge Generator releases prior to 4.0. It also assumes some knowledge of ENVY.

This document is intended for use by team leads, Cross System Product or VisualAge Generator administrators, or other developers responsible for migrating applications to VisualAge Generator 4.0.

Note: Before you migrate, you might want to contact your local IBM representative to learn more about VisualAge Generator service offerings and how they can help you with migration.

How to Use This Document

This document is organized according to migration scenarios. Each part includes all the information needed to migrate from a previous product to VisualAge Generator 4.0. You only need to read the part that applies to your current development product and your choice of VisualAge Generator 4.0 platform. It is recommended that you read the migration considerations and pre-migration checklist before starting your migration.

Terminology used in this document

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

- MVS CICS applies to Customer Information Control System/Enterprise Systems Architecture (CICS/ESA) systems.
- CICS applies to CICS for VSE/ESA, CICS/ESA, CICS for OS/2, CICS for AIX, CICS for Windows NT, and CICS for Solaris.
- CICS for Windows NT refers to IBM TXSeries for Windows NT Version 4.2.
- CICS for AIX refers to IBM TXSeries for AIX Version 4.2.
- CICS for Solaris refers to IBM WebSphere Enterprise Edition Version 3.0.
- IMS/VS applies to Information Management System/Enterprise System Architecture (IMS/ESA) and IMS/ESA Transaction Manager systems.

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

Terminology differences between Java and Smalltalk

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

Table 1. Terminology differences between Java and Smalltalk

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

Documentation provided with VisualAge Generator

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

- Printed and separately ordered using the individual form number.
- Printed and ordered as a set using the bill of forms number or the document kit 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/vagen>).

The following books are shipped with the VisualAge Generator CD and are included in the set you order when you use the single bill of forms (SBOF-6728):

- *VisualAge Generator Getting Started* (GH23-0258-00) ¹
- *VisualAge Generator Installation Guide* (GH23-0257-00) ¹
- *Introducing VisualAge Generator Templates* (GH23-0272-00)

The following documents can be ordered as a set using the bill of forms number (SBOF-6728) or the document kit number, or separately using the individual order numbers.

- *VisualAge Generator User's Guide* (SH23-0268-00) ¹
- *VisualAge Generator Design Guide* (SH23-0264-00) ¹
- *VisualAge Generator Client/Server Communications Guide* (SH23-0261-00) ¹
- *VisualAge Generator Generation Guide* (SH23-0263-00) ¹
- *VisualAge Generator Messages and Problem Determination Guide* (GH23-0260-00) ¹

1. These documents are available as HTML files and PDF files on the product CD.

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

- *VisualAge Generator Server Guide for Workstation Platforms* (SH23-0266-00) ^{1,2}
- *VisualAge Generator Programmer's Reference* (SH23-0262-00) ¹
- *VisualAge Generator System Development Guide* (SG24-5467-00) ¹
- *VisualAge Generator Migration Guide* (SH23-0267-00) ¹
- *VisualAge Generator Templates User's Guide - Standard Functions* (SH23-0269-00)

The following hardcopy documents are available in printed form for VisualGen Host Services for OS/400 and VisualAge Generator Server for MVS, VSE, and VM:

- *Running VisualGen Applications on OS/400* (SH23-6549-01)
- *VisualAge Generator Server Guide for MVS, VSE, and VM* (SH23-0256-00)

The following information is also available for VisualAge Generator:

- *VisualAge Generator External Source Format Reference* (SH23-0265-00)
- *Migrating Cross System Product Applications to VisualAge Generator* (SH23-0244-01)
- *VisualAge Generator Templates on Java—Reference Guide* (SH23-0271-00)
- *VisualAge Generator Templates on Smalltalk—Reference Guide* (SH23-0276-00)

Part 1. Migrating from VAGen 3.x to VAGen 4.0 on Java

Chapter 1. General migration considerations

for VAGen 3.x to Java	3
Migration paths	3
Overview of the V3 to V4 Migration Tool on Java	4
Automatic conversions during 4.0 migration	4
Migrating GUIs	5
Migrating applications, subapplications, and configuration maps	6
Migrating VAGen Templates	7
Establishing naming conventions	8
Assigning ownership	10
Storing control information	10
Generation options.	11
/PROJECT generation option	11
Project List Part	11
Dual maintenance	12
Migrating from OS/2 to Windows NT	12

Chapter 2. Pre-migration checklist 15

Chapter 3. Using the V3 to V4 Migration Tool to migrate VAGen 3.x code and VAGen Templates parts to Java 17

Setting V3 to V4 Migration Tool options on Java	18
Selection Criteria options.	19
Java Naming Convention option	21
Smalltalk Subapplications options.	22
Library Management options	23
Ownership options.	24
Selecting and migrating applications and configuration maps	24
Working with the status log.	26
Resetting the V3 to V4 Migration window	28
Resetting Migration Status Information	29

Chapter 4. Using VAGen Import to migrate VAGen 3.x non-GUI code to Java 31

Chapter 5. Completing the ENVY setup on Java 33

Chapter 6. Completing your migration on

Java	35
Defining control information	35
Generating programs	35
Importing work-in-progress.	35
Recreating ITF resource association information	36
Converting an RTABLE to a Linkage Table.	36

Chapter 1. General migration considerations for VAGen 3.x to Java

Consider the following if you are migrating from VisualAge Generator 3.x to VisualAge Generator 4.0 on Java:

- “Migration paths”
- “Overview of the V3 to V4 Migration Tool on Java” on page 4
- “Automatic conversions during 4.0 migration” on page 4
- “Migrating GUIs” on page 5
- “Migrating applications, subapplications, and configuration maps” on page 6
- “Migrating VAGen Templates” on page 7
- “Establishing naming conventions” on page 8
- “Assigning ownership” on page 10
- “Storing control information” on page 10
- “Generation options” on page 11
- “Dual maintenance” on page 12
- “Migrating from OS/2 to Windows NT” on page 12

Also see “Chapter 2. Pre-migration checklist” on page 15 before you begin to migrate code to VisualAge Generator 4.0.

Migration paths

Depending on your current platform and whether you want to migrate to Smalltalk, Java, or both, different migration paths are available and different considerations apply.

Table 2 on page 4 gives a brief overview of the migration options available for the Java platform. See the referenced chapters for step-by-step procedures for each migration option.

Table 2. Migration Options

Migrating from	Tools to Use	Details on Using the Tools
VisualAge Generator 3.x (non-GUI)	<ul style="list-style-type: none"> V3 to V4 Migration Tool VAGen Import 	<ul style="list-style-type: none"> “Chapter 3. Using the V3 to V4 Migration Tool to migrate VAGen 3.x code and VAGen Templates parts to Java” on page 17 “Chapter 4. Using VAGen Import to migrate VAGen 3.x non-GUI code to Java” on page 31
VisualAge Generator Templates 3.x (non-GUI)	V3 to V4 Migration Tool	“Chapter 3. Using the V3 to V4 Migration Tool to migrate VAGen 3.x code and VAGen Templates parts to Java” on page 17
VisualAge Generator 3.x (GUI)	No tool available	Recreate parts on Java platform
VisualAge Generator 4.0 on Smalltalk parts (non-GUI)	VAGen Import	“Chapter 34. Sharing VAGen 4.0 parts between Java and Smalltalk” on page 351

Overview of the V3 to V4 Migration Tool on Java

The V3 to V4 Migration Tool on Java does the following:

- Migrates versioned configuration maps or applications from VisualAge Generator 3.x on Smalltalk
- Sets flags in the 3.x Smalltalk library to indicate what has been migrated
- Provides special handling for required maps, prerequisite applications, and subapplications because these objects have no corresponding object in Java
- Provides options to control configuration map or application selections, project and package naming conventions, and version, release, and ownership information.

Automatic conversions during 4.0 migration

VisualAge Generator 4.0 automatically makes the following changes to your applications during migration:

- Process and statement group parts are converted to function parts.
- References to processes and statement groups are converted to the new syntax requirements for functions with no parameters.

- PERFORM statements and Unconditional Branch statements are no longer supported and are migrated to Function Invocation statements.
- Subscript parentheses are changed to brackets in VisualAge Generator item names in the following places:
 - 4GL statements in functions (processes and statement groups)
 - Host variable names in SQL statements
 - Comparison value item in DL/I specifications
 - EZEDLPCB is used in a called parameter list
- Calls to EZE service routines are converted to the corresponding function invocation statement. A statement to set the value of EZEREPLY is also added before the function invocation.
- VisualAge Generator-supplied string and math functions are converted from the CALL statement to a function invocation statement or to an assignment statement that contains the function as the source of the assignment. A statement to set the value of EZEREPLY is also added before the function invocation.

VisualAge Generator Templates 4.0 automatically makes the following changes to your applications during migration:

- **Definition and Generation Parameters** of a VAGT instance are stored in two different VAGTemplates part classes. (In version 3.x, these two types of descriptions are stored in the same VAGTemplates part class.)
- **Definition Extensions** for all VAGT entities are stored in an appropriate VAGTemplates part class. (In version 3.x, they are stored in the same VAGTemplates part class as all the instances of the same VAGT entity type.)

In addition, the names of part classes, control information files, and VisualAge Generator palette parts are changed during migration to VisualAge Generator 4.0:

- Table 20 on page 361 shows the changes made to VAGen part class names during 4.0 migration.
- Table 21 on page 361 shows the changes made to VAGen control information part class names during 4.0 migration.
- Table 22 on page 362 shows the changes made to VisualAge Generator palette parts names during 4.0 migration.
- Table 23 on page 362 shows the changes made to VAGen Templates part class names and repartition during 4.0 migration.

Migrating GUIs

VisualAge Generator 4.0 does not provide a way to migrate existing GUI parts to the Java platform. You need to recreate your GUI parts for Java.

Migrating applications, subapplications, and configuration maps

The basic containers for storing code are different on Java and Smalltalk:

- What the Smalltalk platform refers to as **applications** are called **packages** on Java.
- A VisualAge Generator on Smalltalk **configuration map** is similar to a **project** on Java. Though configuration maps are optional on Smalltalk, every Java package must be assigned to a project.

When you use the V3 to V4 Migration Tool, applications are automatically converted to packages, and you must specify the names to assign to the Java projects created by the tool. Subapplications are always migrated when the containing application is migrated. Subapplications can be placed in separate packages or in the package created for the parent application depending on your migration options. Prerequisite applications and required maps are migrated depending on your migration options and whether there are multiple configuration expressions.

If you do not use the V3 to V4 Migration Tool, you must do the following:

- Plan how to organize your code
- Export external source format files from VisualAge Generator 3.x. Generally it is best to use one .esf file for each package you plan to create.
- Manually create the projects and packages for each .esf file
- Import the .esf files, specifying the package names
- Version and release classes, packages and projects

Before you start migrating, you need to decide on a naming convention, ownership structure, and versioning scheme for the packages and projects that are created during migration. Using the V3 to V4 Migration Tool's Migration Options window, you can set defaults that the V3 to V4 Migration Tool then uses during migration to automatically convert application names, version numbers and ownership assignments.

While it is possible to create a project and package for each existing program, you probably have logically grouped existing applications using prerequisites, subapplications, and configuration maps. With VisualAge Generator on Java, you cannot use prerequisites, subapplications or required maps. Instead, you must ensure that you load all required packages before running a package with prerequisites.

Note: You cannot load two projects at the same time that contain the same package.

With VisualAge Generator on Java, you can either organize your code so that all packages needed to run together are in the same project or you can create

separate projects and then create a list of projects that must run together in a project list part (PLP). For more information about PLPs, see “Generation options” on page 11.

Migrating VAGen Templates

The following table outlines considerations that apply when you select VAGen Templates for migration using the V3 to V4 Migration Tool:

Table 3. VAGen Template Migration to VisualAge Generator 4.0 on Java

VisualAge Generator 3.x	VisualAge Generator 4.0 on Java
Standard Generators	The standard generators from VAGen Templates 3.x are not migrated. They have been rewritten for VisualAge Generator 4.0 on Java to make use of the new functions and to provide Java GUI generators.
Customized Generators	Your customized generators from VAGen Templates 3.x are not migrated. You must reapply your customization to the new standard generators for VisualAge Generator 4.0 on Java.
Specifications	The VAGen Templates specifications are migrated automatically by the V3 to V4 Migration Tool. If you want to make use of any of the new functions in the new generators, you can migrate your specifications and then regenerate using the new generators. For example, you could use the migrated specification for a GUI and regenerate it using the Java GUI generators. In addition, you could generate using a migrated specification and one of the new Web generators.

Table 3. VAGen Template Migration to VisualAge Generator 4.0 on Java (continued)

VisualAge Generator 3.x	VisualAge Generator 4.0 on Java
VAGen Templates-generated 4GL code	The 4GL code generated by VAGen Templates 3.x migrates like any other 4GL code. The VAGen Templates generated components include “traceability” information that is used only by VAGen Templates. This information is migrated automatically by the V3 to V4 Migration Tool. After migrating, the 4GL code is ready to be used. Your custom business logic is already incorporated and reacts as a brand new one. This means that you just need to generate the 4GL code using the ‘normal’ generation option. (The ‘override’ option would erase your custom business logic.)
VAGen Templates-generated GUI code	The GUIs generated by VAGen Templates 3.x cannot be migrated. However, you could use the migrated specification for a GUI and regenerate it using the Java GUI generators. You would then need to create the Java equivalent of any Smalltalk business logic you had written in version 3.x. Custom business logic will need to be reapplied after generation with the new 4.0 templates.

Establishing naming conventions

You probably already have naming conventions for parts like processes, records, data items, and so on that you are migrating from VisualAge Generator 3.x. You can continue to use your existing naming conventions, because VisualAge Generator 4.0 retains existing part names during migrations from VisualAge Generator 3.x.

However, VisualAge Generator does change some existing part *types*, as well as making some syntax conversions. For example, process parts and statement group parts are both changed to function parts. Therefore, you might want to establish a naming convention for function parts that is similar to your conventions for processes and statement groups.

See “Automatic conversions during 4.0 migration” on page 4 for a list of changes that VisualAge Generator makes to your existing code during migration.

When you use the V3 to V4 Migration Tool to migrate applications with part types that are changed during migration, you can decide whether the migrated projects and packages are automatically versioned and released during migration. See “Setting V3 to V4 Migration Tool options on Java” on page 18 for more information.

You need to establish naming conventions for the new package and project objects that you must create in Java for your migrated code. Your naming conventions might include the following:

- A unique prefix (possibly three characters) so that all of your packages will be grouped together in the VisualAge on Java workspace. For example, you might choose to prefix all your packages with a company prefix of *xyz*.

If more than one grouping of packages exists within a single project, each group of packages could have a different secondary prefix. You can use Java dot notation to include your global prefix as well as the secondary prefix for each group.

For example, if you have an Accounting system and a Payroll system, you could use your company prefix of *xyz* and then add a secondary prefix of *acct* for the Accounting system and *pay* for the Payroll system. When your packages are all loaded into your VisualAge workspace, they would appear in this order:

```
xyz.acct.pkg1
xyz.acct.pkg2
xyz.pay.pkg1
xyz.pay.pkg2
```

As an alternative, you could leave off the company prefix and create first-level prefixes that describe the tasks performed by the packages. Using prefixes that start with different letters can be an advantage, because some VisualAge windows allow you to enter one letter to quickly tab to the group of packages that start with that letter.

- Unique **version names** to distinguish the existing applications from the migrated packages. You can use the V3 to V4 Migration Tool to set defaults for how the migrated packages will be versioned during migration. See “Setting V3 to V4 Migration Tool options on Java” on page 18 for more information.

In addition to your company-unique naming conventions, there are some general Java naming conventions that you need to follow:

- Java expects package names to be in lowercase characters. When you select the **Java naming convention** check box in the Options window of the V3 to V4 Migration Tool, the tool automatically converts application names to all lowercase characters.

When your migration selections are displayed in the V3 to V4 Migration window, you can change any of the proposed package names created by

the V3 to V4 Migration Tool by typing over the name. If you change a package name to include any uppercase letters, you will get a warning message the first time the V3 to V4 Migration Tool begins to migrate a package with a mixed-case name. However, you can manually override the error message and use a mixed-case name without further errors. If you check the box **Do not show this message again** in the warning window, the warning is only shown for the first migrated package with a mixed-case name.

- Java uses “dot notation” for package names. As part of the conversion to Java conventions, when you select the **Java naming convention** check box in the Options window, the V3 to V4 Migration Tool adds a period to the left of each character (except the first) that it changes from uppercase to lowercase during the migration. For example, if an existing application is named XYZpayrollAppl, the converted package is named x.y.zpayroll.appl. Some common acronyms are automatically preserved as acronyms, with only one period at the end of the acronym. The acronyms that the V3 to V4 Migration Tool preserves are:

AIX	HP	RACF
CICS	IBM	SAA
COBOL	IMS	SQL
CSP	ITF	ST
DB	MSL	US
DLI	MVS	VM
ENVY	OS	VTAM
EZE	PROFS	

For example, if an existing application is named CSPwidgetClose, the converted package is named csp.widget.close. If an existing application is named USBpayApp, the converted package is named us.bpay.app.

Assigning ownership

Using the V3 to V4 Migration Tool, you can decide whether you want to retain the existing ownership structure for applications and configuration maps. When you set the V3 to V4 Migration Tool options, you can choose to maintain the existing ownership structure or change the ownership to the current user. See “Ownership options” on page 24 for more information.

Storing control information

Control information consists of generation option, linkage table, resource association, bind control, and link edit command parts. In VisualAge Generator 3.x, the control information is stored in ENVY parts. When you migrate to 4.0 on Java, the control information is preserved.

Generation options

VisualAge Generator 3.x generation option parts are migrated like all other VisualAge Generator 3.x parts. No special considerations apply.

/PROJECT generation option

A new `/PROJECT` option for the `GENERATE` command is available for VisualAge Generator 4.0 on Java. Using `/PROJECT`, you can generate C++ and COBOL from VisualAge Generator parts stored in the Java repository. Multiple `/PROJECT` options can be specified on the same `GENERATE` command. For example, this command causes both My Project and Your Project to be loaded:

```
hptcmd generate MYPROG /PROJECT="My Project","Version 1.0"  
/PROJECT="Your Project","Version 29.5" ...
```

Note: You cannot use `/PROJECT` for two projects specified on the same `GENERATE` command that contain packages with the same name.

See the *VisualAge Generator Generation Guide* for more information about the `/PROJECT` option.

Project List Part

VisualAge Generator on Java also offers a new project list part (PLP). The PLP is a generation options part in which you specify a list of projects to be loaded prior to loading the project containing the PLP. You can specify the list of projects once in the PLP part and maintain the list in only this one place. The PLP eliminates the need to specify all related projects each time the projects are loaded. You can create a PLP for each group of projects that you want to load together. The PLP is a replacement for required maps in VisualAge Generator on Smalltalk.

When you select the **Select with required maps** check box on the Migration Options window and then select a top-level configuration map for migration, the V3 to V4 Migration Tool creates PLPs automatically for each corresponding project that has required projects. For example, suppose you have the following scenario:

- The top-level configuration map is named *map1*, and it requires *map2*.
- *map2* requires *map3*.

When you select *map1* for migration after setting the **Select with required maps** option, the V3 to V4 Migration Tool loads all three configuration maps into the V3 to V4 Migration window. When you then select the **Migrate** button, the tool migrates the three configuration maps into projects *map1*, *map2*, and *map3* respectively.

During migration, the V3 to V4 Migration Tool creates a default package in *map1* with a generation options part that contains the statement

/PROJECT="map2". The V3 to V4 Migration Tool also creates a default package in *map2* with a PLP options part that contains the statement /PROJECT="map3". When *map3* is created, there is no PLP because there are no required maps for *map3* — it is the end node of the old map structure.

If you select **Auto version and release** on the Migration Options window, the version name is added as a second parameter in each /PROJECT entry of the PLP.

See “Creating a Project List Part (PLP)” on page 154 and the *VisualAge Generator User’s Guide* for information on how to create project list parts.

Dual maintenance

The external source format file for 4GL parts that you export from VisualAge Generator 4.0 is not compatible with the external source format file for VisualAge Generator 3.x. Therefore, if you migrate a subsystem that shares common parts with a subsystem that you will migrate at a later time, you have the following alternatives for maintenance of the common parts:

1. Maintain the common parts in VisualAge Generator 3.x, and when you are satisfied with the changes:
 - a. Export an external source format file from the 3.x application for the changed parts.
 - b. Import the external source format file into VisualAge Generator 4.0 on Java using the **Defined package** radio button so the changes will go into the same package in which the parts are already located.
2. Make the same changes to both the parts in VisualAge Generator 4.0 on Java using VisualAge Generator 4.0 and to the corresponding 3.x parts using VisualAge Generator 3.x.

For GUIs (views), you must make the changes in the corresponding views in VisualAge Generator 4.0 on Java and in VisualAge Generator 3.x on Smalltalk. This is because GUIs cannot be migrated from Smalltalk to Java.

Migrating from OS/2 to Windows NT

Note: If you use a DBCS code page, skip this section. Code page conversion is not required for DBCS code pages.

If you are changing from the OS/2 to the Windows NT development platform, do the following:

- Using VisualAge Generator 3.x on the OS/2 development platform, create a test application containing one process or statement group. In this process or statement group, as comment lines, include all the special characters that you use. Version the test application.
- Migrate the test application using the V3 to V4 Migration Tool to the Windows NT development platform.
- If all the special characters were transferred correctly, you can use the V3 to V4 Migration Tool to do your migration.
- If some of the special characters were not transferred correctly, you must use VAGen Import to migrate your code. See the following sections:
 - “Chapter 4. Using VAGen Import to migrate VAGen 3.x non-GUI code to Java” on page 31
 - “Changing from OS/2 to Windows NT” on page 119

If you plan to use VAGen Import to migrate your code, you can run a similar test using VAGen Import instead of the V3 to V4 Migration Tool to determine whether you can skip the code page conversion step described in “Changing from OS/2 to Windows NT” on page 119.

Chapter 2. Pre-migration checklist

1. Before you migrate, you should first read the following:
 - “Chapter 1. General migration considerations for VAGen 3.x to Java” on page 3
 - “Chapter 3. Using the V3 to V4 Migration Tool to migrate VAGen 3.x code and VAGen Templates parts to Java” on page 17
 - VisualAge Generator 4.0 readme file
2. Save a clean copy of your VisualAge for Java workspace. This clean copy should not contain any of your application code. Copy *ide.icx* to *ideclean.icx*, and store the clean copy on your LAN so that all developers have access to it. Saving copies of the *hpt.ini*, *hptvgj40.ini*, *ide.ini*, and *ivj.dat* files is also recommended.
3. Check with IBM support to see if there are any fixes available for the VisualAge Generator 4.0 V3 to V4 Migration Tool.
4. Contact your local IBM representative to learn more about VisualAge Generator service offerings that can help you with migration.
5. Be sure that the configuration maps and applications that you plan to migrate have been versioned on VisualAge Generator 3.x.

Chapter 3. Using the V3 to V4 Migration Tool to migrate VAGen 3.x code and VAGen Templates parts to Java

VisualAge Generator 4.0 on Java includes a **V3 to V4 Migration Tool** for migrating 3.x code and VAGen Templates parts to 4.0. The V3 to V4 Migration Tool allows you to migrate applications and configuration maps from your 3.x ENVY library to VisualAge Generator 4.0 on Java, using options that apply across migration operations. You can change the options as needed. For example, you can set options that apply for one group of applications and configuration maps, migrate that group, and then set different options for the next group to be migrated.

Note: The V3 to V4 Migration Tool migrates only non-GUI code to VisualAge Generator 4.0 on Java. Because GUI migration to Java is not supported, GUI parts are ignored when applications are selected for migration.

Use these steps to start the V3 to V4 Migration Tool:

1. From the IBM VisualAge Generator 4.0 folder, select **VAGen Developer 4.0 on Java with Migration**.
2. From the VisualAge for Java workspace, select **Workspace->Open VAGen Parts Browser**. The VAGen Parts Browser window is displayed.
3. From the VAGen Parts Browser window, select **Tools->Migration->V3 to V4 Migration**. The main V3 to V4 Migration Tool window, called V3 to V4 Migration, is displayed.

Figure 1 shows the V3 to V4 Migration window.

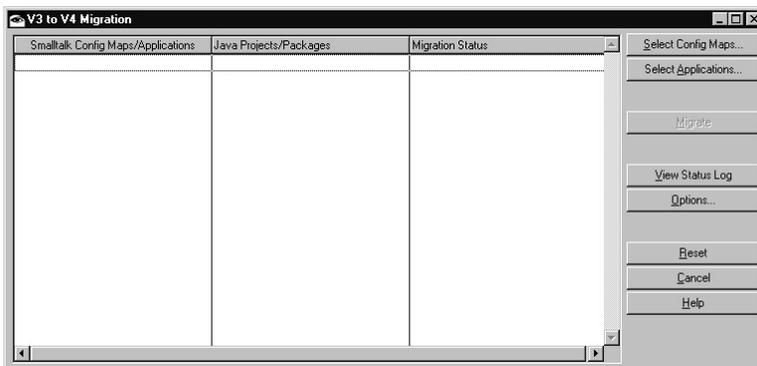


Figure 1. V3 to V4 Migration window

Note: You must start VisualAge Generator 4.0 on Java using the **VAGen Developer 4.0 on Java with Migration** option for the migration tools to be loaded in your workspace. After you have finished migrating all your 3.x code, you can start VisualAge Generator 4.0 without the migration tools to save memory and increase performance.

You can also start VisualAge Generator 4.0 with the V3 to V4 Migration Tool from a command line, using these steps:

1. Open a command line window to the directory where the VisualAge Generator 4.0 ide.icx file is stored, and enter the command `ide /vgmig`. (Note the blank space after **ide**.)
2. From the Log window, select **Workspace->Open VAGen Parts Browser**. The VAGen Parts Browser window is displayed.
3. From the VAGen Parts Browser window, select **Tools->Migration->V3 to V4 Migration**. The main V3 to V4 Migration Tool window, called V3 to V4 Migration, is displayed.

From the V3 to V4 Migration window, you can perform the following tasks:

- Set migration options
- Select and migrate configuration maps and applications
- View and work with the status log
- Reset the V3 to V4 Migration window

These tasks are described in the following sections.

Setting V3 to V4 Migration Tool options on Java

The V3 to V4 Migration Tool allows you to set migration options that can be applied for all applications and configuration maps that you later select for migration.

When you select applications and configuration maps to be migrated to VisualAge Generator 4.0, the V3 to V4 Migration Tool uses some of the options you set in the Migration Options window to display the applications and configuration maps that are available for migration. Other options are used during the migration for versioning and ownership assignment. You can migrate one group of applications or configuration maps using one set of options, and then change the options before migrating another group.

To access the Migration Options window from the V3 to V4 Migration window, select the **Options** push button.

Figure 2 on page 19 shows the Migration Options window.

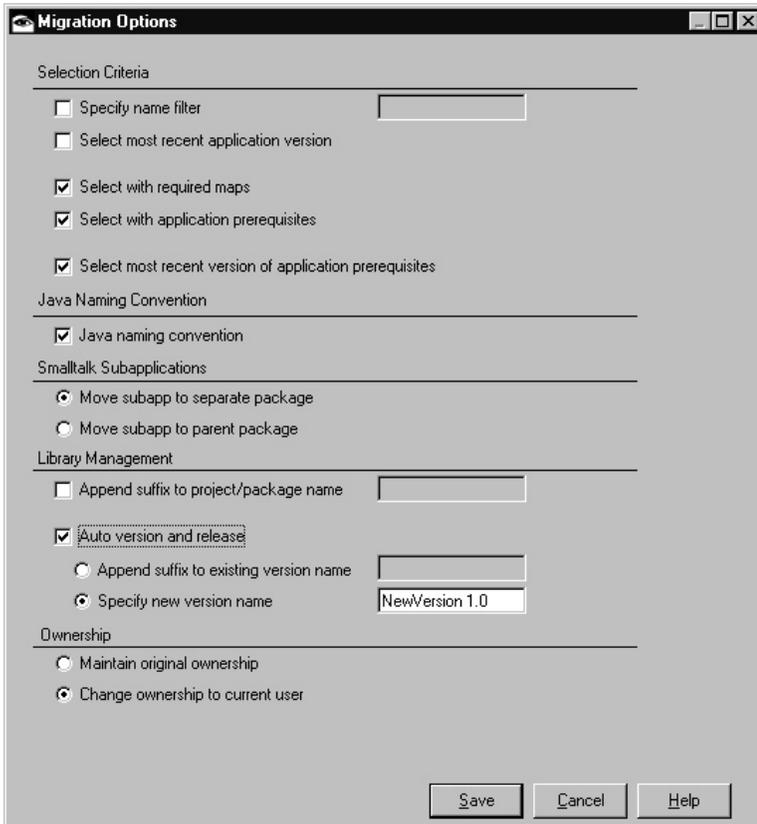


Figure 2. Migration Options window

In the Migration Options window, you can set defaults that are applied to all applications and configuration maps you later select for migration. The Migration Options window is divided into groups of options. These option groups are described in the following sections.

Selection Criteria options

The Selection Criteria options allow you to specify options that aid in application and configuration map selection when you later select **Select Applications** or **Select Config Maps** from the V3 to V4 Migration window.

Table 4. Selection criteria migration options

Option	Migration processing
Specify name filter	Allows you to use a naming convention to select a group of applications or configuration maps for migration. When this box is checked, the Filter prompter window is displayed each time you later select the Select Applications and Select Config Maps push buttons on the V3 to V4 Migration window.
Select most recent application version	With this box checked, when you later select an application for migration, the most recent version is automatically selected for you.
Select with required maps	<p>With this box checked, when you later select a configuration map for migration, all the required configuration maps for your selected configuration map are automatically loaded into the V3 to V4 Migration Tool along with your selected configuration map.</p> <p>Note: If a configuration map has more than one configuration expression, VisualAge Generator 4.0 on Java cannot determine which configuration expression to use. In this case, no required maps are loaded and a warning message is written to the V3 to V4 Migration Tool's status log.</p>

Table 4. Selection criteria migration options (continued)

Option	Migration processing
Select with application prerequisites	<p>With this box checked, when you later select an application for migration, all the user-created prerequisite applications that could contain VAGen parts are automatically loaded into the V3 to V4 Migration Tool along with your selected application. If more than one version of a prerequisite application is available, when you later select an application for migration, you will be prompted to select the version you want.</p> <p>Note: If an application has more than one configuration expression, VisualAge Generator 4.0 on Java cannot determine which configuration expression to use. In this case, no prerequisite applications are loaded and a warning message is written to the V3 to V4 Migration Tool's status log.</p>
Select most recent version of application prerequisites	<p>With this box checked, when you later select an application for migration, the most recent version of each prerequisite for that application is automatically loaded into the V3 to V4 Migration Tool along with your selected application.</p>

Java Naming Convention option

When you check the box for the **Java Naming Convention** option, the following conversions are performed:

- **When applications are converted to packages, the names are changed to all-lowercase characters.** Java naming conventions expect package names to be in all-lowercase characters. If you do not check this box, you will get a warning message the first time the V3 to V4 Migration Tool begins to migrate an application with a mixed-case name. However, you can manually override the error message and use a mixed-case name without further errors. If you check the box "Do not show this message again" in the warning window, the warning is only shown for the first migrated application.
- **When applications are converted to packages, the names are converted to dot notation.** Java naming conventions use "dot notation" for package names. As part of the conversion to Java conventions, the V3 to V4 Migration Tool adds a period to the left of each character (except the first) that it changes to lowercase during the migration. For example, if an

existing application is named XYZpayrollAppl, the converted package is named x.y.zpayroll.appl when you check this box.

Some common acronyms are automatically preserved as acronyms, with only one period at the end of the acronym. The acronyms that the V3 to V4 Migration Tool preserves are:

AIX	HP	RACF
CICS	IBM	SAA
COBOL	IMS	SQL
CSP	ITF	ST
DB	MSL	US
DLI	MVS	VM
ENVY	OS	VTAM
EZE	PROFS	

For example, if an existing application is named CSPwidgetClose, the converted package is named csp.widget.close when you check this box. If an existing application is named USBpayApp, the converted package is named us.bpay.app.

Smalltalk Subapplications options

The Java platform does not support the concept of subapplications. You cannot create "subpackages" on Java. Therefore, you must decide how to reorganize subapplications when you migrate them to VisualAge Generator 4.0 on Java.

The Smalltalk Subapplications options allow you to specify whether subapplications will be migrated into the same package as the parent application or a different package when you later select applications for migration that include subapplications.

Note: If an application has more than one configuration expression, VisualAge Generator 4.0 on Java cannot determine which configuration expression to use. In this case, all possible subapplications are migrated with the parent application.

Table 5. Smalltalk subapplications migration options

Option	Migration processing
Move subapp to separate package	Each subapplication is migrated to a separate package from its parent application. The separate package is included in the same project as the package that corresponds to the parent application.
Move subapp to parent package	The VAGen parts in each subapplication are added to the same package as the parent application.

Library Management options

The Library Management options allow you to specify naming conventions for migrated configuration maps, applications, and classes. This group of options also enables you to specify versioning and releasing conventions for migration.

Table 6. Library management migration options

Option	Migration processing
Append suffix to project/package name	Adds a suffix to the end of each existing application or configuration map name as it is migrated to a package or project. In the text box to the right of this option, you can specify the suffix to be appended.
Auto version and release	<p>All configuration maps, applications, and classes are automatically versioned and released during migration.</p> <p>When you check this box, you can also select Append suffix to existing version name or Specify new version name.</p> <p>If you do not check this option, all editions of migrated projects, packages, and classes are left open, and developers and owners must version and release their own projects, packages, and classes after migration. Also, if you do not check this box, you cannot select either of the sub-options under this option.</p>
Append suffix to existing version name	Adds a suffix to the existing version name. In the text box to the right of this option, you can specify the suffix to be appended. Using a suffix can help in associating the migrated version with the original version and in determining which other versions you still need to migrate. If you check this box but do not specify a suffix to append, the error message <code>Text required if selected</code> is displayed.
Specify new version name	Enables you to specify a new version name to be assigned for all your migrated projects, packages, and classes. If you check this box but do not specify a version name, the error message <code>Text required if selected</code> is displayed.

Ownership options

The Ownership options allow you to specify whether the ownership assignments for your current configuration maps, applications, and classes will be preserved during migration or changed.

Note: Regardless of the Ownership option you select, you can select the **Auto version and release** option under Library Management.

Table 7. Ownership migration options

Option	Migration processing
Maintain original ownership	Allows you to maintain the existing ownership structure for all migrated projects, packages, and classes.
Change ownership to current user	Assigns ownership for all migrated projects, packages, and classes to the user who performs the migrations. After the migration process completes, you can change the ownership back to the original owner or any other owner.

Selecting and migrating applications and configuration maps

Follow these steps to migrate applications and configuration maps using the V3 to V4 Migration Tool:

Note: You cannot use the V3 to V4 Migration Tool to migrate an open edition of a 3.x application or configuration map. You must use VisualAge Generator 3.x to version and release each application and configuration map that you want to migrate to VisualAge Generator 4.0.

1. Open the VAGen Parts Browser and then select **Tools->Migration->V3 to V4 Migration** to start the V3 to V4 Migration Tool.
2. Select the **Options** push button to set migration preferences that will be used during the migration process. (Some options can be changed again before you select the **Migrate** push button.)
3. Display a list of applications or configuration maps available for migration by selecting one of the following:
 - Select the **Select Applications** push button if you want to migrate applications.
 - Select the **Select Config Maps** push button if you want to migrate configuration maps.

Note: You cannot select a mixture of configuration maps and individual applications for migration at the same time. You can migrate a

group of applications or a group of configuration maps, but not both together. When you select one of these buttons, the other button is disabled until you select the **Reset** button. When you select **Reset**, the list of selections currently in the V3 to V4 Migration window is deleted, and both **Select Applications** and **Select Config Maps** are available for selection again.

4. In the window requesting the IP address or host name running the server, enter the name of the machine where your 3.x ENVY library resides, and select **OK**.
5. In the window requesting the full path name of the library, enter the directory and file name of your 3.x ENVY library, and select **Open**.
6. If you selected the **Specify name filter** option on the Migration Options window, the Filter prompter window is displayed. Enter a case-sensitive prefix to be used in filtering the application list. For example, if you want to see only applications that begin with Xyz, such as XyzPayApp and XyzEmployeeApp, enter **Xyz** in this window. You can use the * (asterisk) wildcard in specifying the search string. For example, use **Xyz*Sample*** to find applications such as Xyz03SampleApp, XyzMMediaSampleApp, and XyzPaySampleOpnsApp.
7. On the Selection Required window, select an application or configuration map and a version to be migrated, and then select the >> arrow button to add your selection to the Selected Versions column.

After you have selected a version for each application or configuration map that you want to migrate, select **OK**.

For applications only, if you selected the **Select most recent application version** option under the Selection Criteria section of the Migration Options window, this window shows only a single-column list of applications, because the versions have been selected for you. You can select multiple applications for migration. When you are finished selecting applications, select **OK**.

8. For applications only, the Java Project name prompter window is displayed next. Enter the name of a new project in which the migrated packages for your selected applications will be stored, or enter the name of an existing project, and select **OK**. If you enter the name of a new project, the V3 to V4 Migration Tool creates the project for you. If you enter the name of an existing project, the V3 to V4 Migration Tool opens a new edition of the project if an edition is not already open.
9. A populated version of the V3 to V4 Migration window is displayed. It lists the applications and configuration maps you selected for migration as well as the project and package names that will be used to store the migrated applications and configuration maps.

Before migrating, you can add more applications or configuration maps to this list, using the above steps.

You can delete any application or configuration map from the list to be migrated by selecting it in the Smalltalk Config Maps/Applications column and then selecting **Remove selected** from the context menu for that column. You cannot delete applications within a selected configuration map.

You can change the project and package names in the second column of this window by typing over the names in the window or by using the **Find/Replace** option in the context menu for the second column. Find/Replace works on all selected applications and configuration maps. You can select **Select All** before selecting **Find/Replace** to make a change to all names listed in the V3 to V4 Migration window. The string you enter in the Find/Replace prompter window can be a full name or a partial name, such as a prefix.

10. When you are ready to migrate the listed applications and configuration maps, select the **Migrate** push button. All applications and configuration maps listed in the window are migrated. Flags are set in the Smalltalk library to indicate which applications and configuration maps have been migrated. The current settings for the Smalltalk Subapplications, Library Management, and Ownership migration options are used for all of the applications or configuration maps that are being migrated.

During migration, the V3 to V4 Migration window is refreshed with status information in the Migration Status column. When all the applications and configuration maps have been successfully migrated, the Migration Status column shows a status of **Migrated** for each application and configuration map. The status information is also written to the status log. If an application or configuration map shows a status of **Migration Error...check status log**, you can check the status log to see which step in the migration process was the last to complete successfully for that application.

Working with the status log

The V3 to V4 Migration Tool stores a record of each step in the migration process in a status log. To view the status log, from the V3 to V4 Migration window, select the **View Status Log** push button.

Figure 3 on page 27 shows the Migration Status Log window.

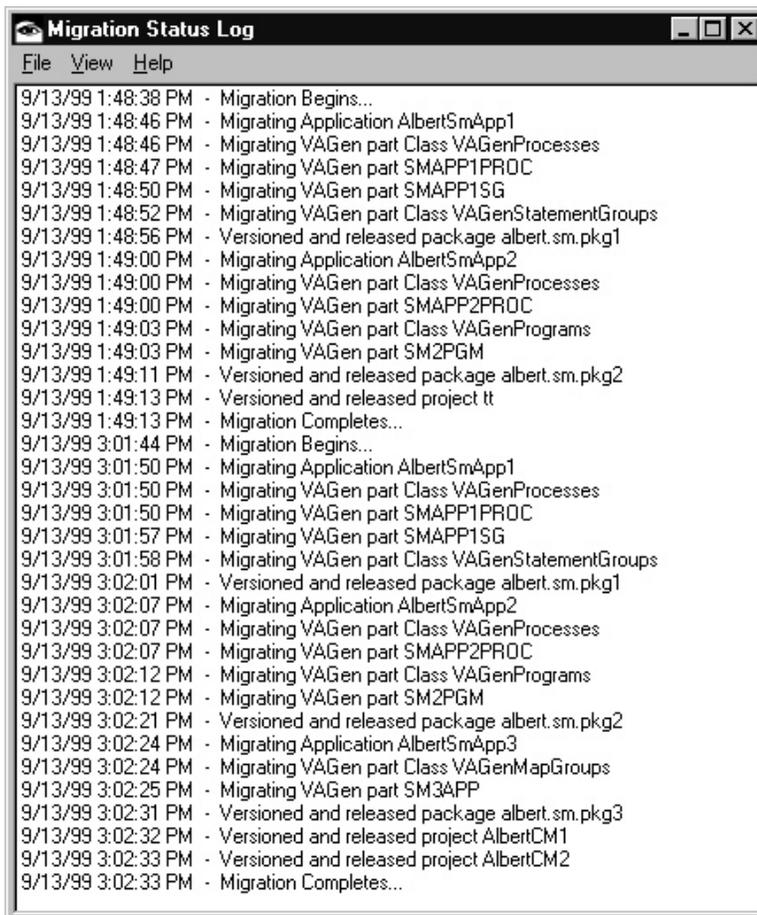


Figure 3. Migration Status Log window

The status log is stored in the file **mgstatus.log**. This log is cumulative. The information in this log is only deleted when you select **File->Clear** from the menu in the Status Log window.

If you have hundreds of applications and configuration maps to migrate, the status log can become quite large. At some point, you might want to clear the status log of the information from previous migration operations before you begin migrating the next group of applications and configuration maps.

Before clearing the status log, however, you might want to print it (by selecting **File->Print**) or copy it to another file (by selecting **File->Save as**). When you use **Save as** to save the status log to another file, if that other file exists, a prompt appears asking if you want to overwrite the file or append to it.

To clear the status log, select **File->Clear** from the menu in the Status Log window.

So that you do not have to search the entire, cumulative log to find information of interest to you, several filtering options are available that allow you to view selected portions of the status log:

All	Select this menu choice to view the entire status log contents.
Applications	Select this menu choice to view only the information about applications that have been migrated.
Errors	Select this menu choice to view only the error messages in the log.
Information	Select this menu choice to view only the information messages in the log.
Warnings	Select this menu choice to view only the warning messages in the log.
Search for string	Select this menu choice to search for a text string in the status log. Enter your search string in the dialog that is displayed after you select this option. The search string is case-sensitive.

For more information about the error, information, and warning messages displayed in the status log, see the *VisualAge Generator Messages and Problem Determination Guide*.

Resetting the V3 to V4 Migration window

The V3 to V4 Migration Tool can keep information in the V3 to V4 Migration window on all completed migration operations. As each new migration is started, the V3 to V4 Migration Tool automatically appends information to the existing contents of the window.

There might be times, however, when you do not want information on already migrated code to remain in the V3 to V4 Migration window while you are migrating the next group of applications and configuration maps. For example, you might have migrated several groups of applications and configuration maps that share common code among them but that do not contain any code shared by your remaining unmigrated applications and configuration maps. In this case, the information on past migrations is not needed for the next migration operation. To remove the unneeded information, you can reset the V3 to V4 Migration window before migrating the next group.

When you reset the V3 to V4 Migration window, all information about past migration operations is deleted from the window. To reset the window, select the **Reset** push button. When you select **Reset**, the information on past migrations is still stored in the migration status log, **mgstatus.log**, and both the **Select Config Maps** and **Select Applications** buttons are enabled again.

Resetting Migration Status Information

The V3 to V4 Migration Tool sets flags in the Smalltalk library to record the migration status of configuration maps and application. There might be times when you want to reset this status. For example, after completing a pilot migration, you might want to reset the migration status before doing the final migration. To reset the status, from the V3 to V4 Migration window, select the configuration maps and applications that you want to reset. Then press mouse button 2 and select **Mark Not Migrated**. The flags in the Smalltalk library are reset to indicate that the selected configuration maps and applications have not been migrated.

Note: Selecting **Mark Not Migrated** does not delete the migrated parts in the Java repository.

Chapter 4. Using VAGen Import to migrate VAGen 3.x non-GUI code to Java

When you use VAGen Import instead of the V3 to V4 Migration Tool to migrate 3.x code to VisualAge Generator 4.0, the following restrictions apply:

- All configuration map information is lost. You will need to create projects from scratch in VisualAge Generator 4.0.
- Existing ownership information is lost. Ownership is set to the userid of whichever user creates the projects and packages and imports the parts.
- The steps in this section assume that you want to migrate each ENVY application to a separate Java package. If you want to reorganize your code, export one .esf file for each Java package that you want to create.
- Do not forget to migrate prerequisite applications and subapplications that might not be loaded into your VisualAge Generator 3.x image based on the configuration expressions used for the parent application.

To migrate VisualAge Generator 3.x non-GUI code to VisualAge Generator 4.0 on the Java platform, use these steps:

1. Start VisualAge Generator 3.x on Smalltalk.
2. Export your existing applications to external source format files. You must use your existing VisualAge Generator 3.x product to create these .esf files.

From the VisualAge Organizer window for VisualAge Generator 3.x, select **Applications->Import/Export->VAGen Export** to create the external source format files. Export one external source format file for each existing ENVY application.

Notes:

- a. If you are migrating from VisualAge Generator 3.x on an OS/2 development platform to VisualAge Generator 4.0 on a Windows NT development platform, see section "Migrating from OS/2 to Windows NT" on page 12 for information on code page conversions.
 - b. You should not modify the export files.
3. Start VisualAge Generator 4.0 on Java.
 4. In VisualAge Generator 4.0, create a Java project and package to store the code you want to import:
 - a. From the VisualAge for Java Workbench window, open the context menu by clicking the right mouse button on an empty part of the workspace.
 - b. Select **Add->Project** from the context menu.

- c. In the Add Project window, enter a name for the new project and select the **Finish** push button.
- d. From the Workbench window, select the project you just created, and open the context menu.
- e. From the context menu, select **Add->Package**.
- f. In the Add Package window, enter a name for the new package and select the **Finish** push button.

Create one Java package for each .esf file that you created in step 2. For more information about creating packages, see the *VisualAge Generator User's Guide*.

5. From the VisualAge for Java Workbench window, select **Workspace->Open VAGen Parts Browser** to open the VAGen Parts Browser.
6. From the VAGen Parts Browser window, select **Parts->Import/Export->VAGen Import**.
7. In the VAGen Import File Selection window, select one external source format (*.esf) file that you want to import and then select the **Open** button.
8. In the VAGen Import window, check to see if there are any parts listed in the Parts with errors list box. If there are, you need to debug the errors before those parts can be imported with the rest of the .esf file. When there are no parts with errors, proceed to the next step.
9. In the VAGen Import window, specify the name of your package in the Target package field. Move all parts to be imported from the Available parts list to the Selected parts list. Then select the **Import** push button. The selected parts are imported into the package you specified.
10. In the VAGen Import window, select the **New File** push button and then repeat steps 7, 8, and 9 for each external source format file you need to import. When you have finished importing your .esf files to VisualAge Generator 4.0, select the **Cancel** push button in the VAGen Import window to return to the VAGen Parts Browser window.
11. Create the default packages and Project List Parts as required based on your planned repository architecture.
12. Version and release all classes, packages and projects for the existing applications you have imported.
13. Save your VisualAge Generator 4.0 workspace.
14. See "Chapter 5. Completing the ENVY setup on Java" on page 33 and "Chapter 6. Completing your migration on Java" on page 35 for more steps you should take to complete the migration.

Chapter 5. Completing the ENVY setup on Java

When you use the V3 to V4 Migration Tool to migrate from VisualAge Generator 3.x to VisualAge Generator 4.0 on Java, all of the ENVY setup tasks are completed for you automatically by the V3 to V4 Migration Tool, provided you select the V3 to V4 Migration Tool option **Auto version and release**. If you do not set this option before using the V3 to V4 Migration Tool to migrate 3.x code, or if you do not use the V3 to V4 Migration Tool to migrate, you will need to version the migrated 4.0 code manually. In that case, see “Chapter 15. Completing the ENVY setup on Java” on page 151 for information on how to complete the setup tasks manually.

Chapter 6. Completing your migration on Java

The following sections describe specific tasks you might need to perform to complete your migration to VisualAge Generator 4.0. These tasks include:

- “Defining control information”
- “Generating programs”
- “Importing work-in-progress”
- “Recreating ITF resource association information” on page 36

Defining control information

Control information that is needed for test and generation must be stored in ENVY packages. This control information consists of:

- Generation options
- Linkage table
- Resource associations
- Bind control information
- Link edit information

When you use the V3 to V4 Migration Tool to migrate your 3.x applications, you can migrate control information to VisualAge Generator 4.0 on Java packages just as you would migrate any other ENVY applications.

Generating programs

After you finish migrating a group of Smalltalk applications to Java packages, you might want to generate the programs to help ensure that you have migrated the correct version of your code.

If you are migrating from VisualAge Generator 3.0, any programs for the C++ target environment **must** be regenerated.

Importing work-in-progress

Suppose you have already migrated and versioned a development system. Then, three weeks after migration, you discover 50 4GL parts have been changed on the VisualAge Generator 3.x development system. The 4GL parts are contained in several applications. You can use the V3 to V4 Migration Tool to migrate the new versions of the changed applications. Be sure to set your migration options to the same settings as in your original migration. The V3 to V4 Migration Tool will create new editions of all 4GL parts from the corresponding parts in the modified applications.

The advantage of using the V3 to V4 Migration Tool is that you only need to identify the applications that must be re-migrated. Another advantage is that migration of this version of the applications is logged in the status log, **mgstatus.log**. The disadvantage is that all the 4GL parts in the changed applications are re-migrated and result in new editions in the Java repository.

Alternatively, you can export an external source format file that contains only the 4GL parts that have been changed. You can then use **VAGen Import** to migrate the parts. See “Importing work-in-progress” on page 161 for information on how to use VAGen Import to migrate your work-in-progress. Using **VAGen Import** requires you to load the affected projects and packages into your workspace. The advantage of using **VAGen Import** is that you only need to migrate the 4GL parts that were changed, rather than all the 4GL parts in the affected applications.

Recreating ITF resource association information

If you use serial, indexed or relative files in the Interactive Test Facility, you use resource association information to point to the files. ITF resource association information is not stored as a resource association part. You need to recreate the ITF resource association information for VisualAge Generator 4.0 on Java.

Converting an RTABLE to a Linkage Table

If you have been using the VisualAge Generator middleware RTABLE for communications routing, the RTABLE entries must be moved from the RTABLE to a linkage table. The following example shows a mapping of RTABLE entries to linkage table entries:

```
RTABLE
app1 - - - - - 1u2 LU2C - 1
app2 - - - - - 1u2 LU2C - 1
app3 - - - - - 1u2 LU2C - 1
app4 - - - - - 1u2 LU2B - 1
$ANY - - - - - 1u2 LU2K - 1

LINKAGE TABLE
:calllink applname=app1 linktype=remote remotecomtype=LU2
serverid=LU2C.
:calllink applname=app2 linktype=remote remotecomtype=LU2
serverid=LU2C.
:calllink applname=app3 linktype=remote remotecomtype=LU2
serverid=LU2C.
:calllink applname=app4 linktype=remote remotecomtype=LU2
serverid=LU2B.
:calllink applname=* linktype=remote remotecomtype=LU2
serverid=LU2K.
```

Part 2. Migrating from VAGen 2.x or Cross System Product to VAGen 4.0 on Java

Chapter 7. Comparing MSLs and library management on VAGen 4.0 on Java

ENVY characteristics	39
Comparison of MSLs and ENVY	41
Member types	41
Storing members	42
Storing control information	42
MSL concatenation	43
Functional organization	44
Member associations	44

Chapter 8. General migration considerations for VAGen 2.x and Cross System Product to Java

Migration paths	47
Automatic conversions during 4.0 migration	48
Migrating GUIs	49
Resolving duplicate member names	49
Establishing naming conventions	50
Organizing your code for ENVY	52
Functional organization	52
Assigning ownership	53
Storing control information	54
Multiple control packages	54
Single control package	55
Generation options	56
/PROJECT generation option	56
Project List Part	57
Dual maintenance	57
Migrating from Cross System Product	57
Migrating from OS/2 to Windows NT	58
Using the migration log file	58

Chapter 9. Pre-migration checklist

Chapter 10. The MSL Migration Assistance Tool on Java

Overview of using the MSL Migration Assistance Tool	62
Building MSL directories	63
Selecting MSLs - using the MSL Library Selection window	64

Selecting parts for a part list - using the Part List Selection Criteria View window	65
Selecting parts to move to ENVY - using the MSL Migration Part List window	67
Special columns in the MSL Migration Part List window	68
Other MSL Migration Part List functions	70
Working in the sandbox - using the VG Part Prerequisites View window	70
Identifying common code	70
Identifying missing parts	71
Other sandbox functions	71
Reloading previously migrated ENVY packages back into the MSL Migration Assistance Tool sandbox	72

Chapter 11. Migrating production and work-in-progress MSLs to VAGen 4.0 on Java

Production MSLs	76
Techniques for moving parts to the sandbox	77
All data items, records, and tables are used	77
All records and tables are used	78
Some records and tables might not be used	78
Considerations for the migration techniques	79
Work-in-progress MSLs	80
Using VAGen Import	81
Using the MSL Migration Assistance Tool	82

Chapter 12. VAGen on Java case studies based on various MSL structures

Understanding the diagrams and terminology	85
MSL structure diagrams	85
Advance command in VisualAge Generator	86
Multiple subsystems with no duplicates	87
Recommendations	87

Multiple subsystems with controlled duplicates	89
Recommendations	89
Separate production MSLs for each developer	92
Recommendations	93
Creating a core common set of packages	93
Creating stand-alone subsystems	95
MSLs that contain unintended duplicates	97
Recommendations	98
MSLs containing code from VisualAge Generator Templates or BW*Wizard	98
Recommendations	100
Special considerations for VisualAge Generator Templates	102
Complete set of MSLs for production and deltas for test	102
Recommendations	104
Complete sets of MSLs for test and production	106
Recommendations	108
MSLs from marketing or other demonstrations.	111
Recommendations.	112
Using a single package	112
Using multiple packages	113
Chapter 13. Running the MSL Migration Assistance Tool on Java	115
Starting VisualAge Generator	115
Creating users and setting the current user	116
Collecting your source code	117
From Cross System Product	117
From VisualAge Generator with TeamConnection and no MSLs	117
From VisualAge Generator MSLs.	117
Handling code page changes	118
Using the HPTRULES.NLS file	118
Changing from OS/2 to Windows NT	119
Starting the MSL Migration Assistance Tool	121
Building MSL directories	121
Resetting the sandbox from ENVY	123
Selecting your MSLs	125
Selecting and migrating VAGen parts	126
Creating a new package.	129
Moving a VAGen part between packages	129
Controlling the creation of package.nodes	130
Renaming a package	131
Collapsing a package	132
Handling Duplicates	132
Controlled Duplicates	133

Unintended Duplicates	133
Duplicates for business logic	136
Finding the package in which a part is located	137
Listing missing (not found) parts	138
Handling missing (not found) parts.	139
Checking relationships among packages	140
Determining which programs are referenced	140
Determining the parts that are referenced	141
Checking consistency of packages	141
Updating the list of required packages.	142
Changing the list of required packages	142
Normalizing the list of required packages	143
Deleting a package.node	143
Deleting a package	144
Deleting one package	144
Deleting all packages	145
Committing to ENVY	145

Chapter 14. Using VAGen Import to migrate VAGen 2.x and Cross System Product non-GUI code to Java. 149

Chapter 15. Completing the ENVY setup on Java 151

Versioning and releasing a VAGen part class	151
Versioning and releasing a package	152
Versioning a project	153
Creating a Project List Part (PLP)	154
Changing the owner of a project.	155
Assigning ownership of a VAGen part class	155
Adding group members	155
Changing the ownership of a VAGen part class	156
Changing the owner of a package	156

Chapter 16. Completing your migration on Java 159

Defining control information	159
Generating Programs	160
Importing work-in-progress	161
Migrating VSAM files	163
Converting an RTABLE to a Linkage Table	163

Chapter 7. Comparing MSLs and library management on VAGen 4.0 on Java

In both Cross System Product and releases of VisualAge Generator prior to 3.0, code was written in small pieces called members. Members were stored in Member Specification Libraries (MSLs). In VisualAge Generator 3.0 or later, the code must be stored in the VisualAge for Java repository.

If you plan to use code from Cross System Product and previous releases of VisualAge Generator, you must migrate the code from the MSLs to ENVY. This chapter explains the following:

- “ENVY characteristics”
- “Comparison of MSLs and ENVY” on page 41

ENVY characteristics

To provide interoperability with VisualAge for Java, VisualAge Generator 4.0 shares the VisualAge for Java library management system. This library management system is called ENVY.

There are new terms that are important for the ENVY environment. The following terms are new for VisualAge Generator 4.0:

New Term

Relationship to MSL Concepts

VAGen part class

Each 4GL member type (all member types except GUIs) becomes a **VAGen part class**. The VAGen part classes created for the member types are prefixed by *VAGen* (for example, *VAGenRecords*).

There are five additional VAGen part classes that are used to contain control information that was stored outside the MSL in previous releases of VisualAge Generator. These VAGen part classes are for linkage table, resource association, generation options, bind control, and linkage editor information.

VAGen part

Each 4GL member is now stored as a **VAGen part**. A VAGen part is associated with a Java **method** in its VAGen part class. The VAGen parts appear in the **VAGen Parts** pane of the

VAGen Parts Browser window when the package and type containing that part are selected.

View

Each GUI is now a view.

Note: VisualAge Generator 2.x GUIs cannot be migrated to VisualAge Generator 4.0 on Java. You must recreate the GUIs as Java classes.

Program

The *application* member type has been changed to **program** to distinguish it from an ENVY application in VisualAge Generator on Smalltalk.

Package

A package is a group of classes and methods that are closely related in function. A package can include VAGen part classes and VAGen parts.

Project

A **project** is a group of package editions that should be loaded together into a developer's workspace.

The following ENVY concepts are new for VisualAge Generator 4.0:

Concept

Description

Functional Organization

ENVY enables you to group parts into packages. These packages can (and should) be organized along functional lines. Similarly, packages are organized into projects. Each project should be a group of functionally related packages.

Ownership

Each project and each package has an assigned owner who is responsible for the integrity of the code that is placed in the project or package.

Each part (class) or VAGen part class has an assigned owner who is responsible for the integrity of the code that is placed in the class. For 4GL parts, this means that the owner of the class *VAGenRecords* within package xyz is responsible for the integrity of **all** record definitions stored as part of package xyz. Because each GUI is a separate view (visual part), each GUI within package xyz can have

a different owner. 4GL parts used within the GUI become VAGen parts.

Note: In releases of VisualAge Generator prior to 3.0, the closest concept to ownership was write-protecting the staging, test, or production MSLs and only giving a team leader the authority to advance members into these MSLs.

Edition

Each change that is made to a 4GL VAGen part results in a new edition of the VAGen part being stored in the ENVY repository. Editions of parts, packages, and projects are also stored in the ENVY repository.

Version

Editions can be frozen to prevent further changes to that level of code. The frozen edition is called a **version**. After a part, package, or project is versioned, the only way to make changes is to open a new edition.

Workspace

A workspace is the developer's current view of the ENVY repository. It contains the version or edition of the projects, packages, and parts that the developer wants to work on.

Comparison of MSLs and ENVY

This section describes how concepts you are familiar with for MSLs relate to concepts in the ENVY library manager.

Member types

In Cross System Product and releases of VisualAge Generator prior to 3.0, there was one member type for each type of code that could be written.

With VisualAge Generator 3.0 and later, each 4GL member type is a VAGen part class that is prefixed with *VAGen* (for example, *VAGenRecords*). For GUIs, there is no corresponding VAGen part class because each GUI is a separate class.

Table 20 on page 361 shows the correspondence between member types and VAGen part classes.

Storing members

In releases of VisualAge Generator prior to 3.0, an MSL was an OS/2 directory and each member was a file within the directory. In Cross System Product, an MSL was a VSAM file and each member was stored as records within the file.

With VisualAge Generator 3.0 and later, all information is stored in the ENVY repository. Each 4GL member is a VAGen part and is associated with a Java method. Each GUI is a view (visual part) and is a Java class.

Storing control information

For Cross System Product and releases of VisualAge Generator prior to 3.0, most control information related to test and generation was stored outside the MSL. This control information included:

- Generation options that indicate how an application is to be generated. For example, generation options control whether working storage records are to be initialized, what high-level qualifier on the host is to be used for preparation, and what linkage table is to be used.

For Cross System Product, only COBOL generation used generation options and these were stored in separate files outside the MSL. For releases of VisualAge Generator prior to 3.0, COBOL generation options were stored in separate files.

- Linkage table that indicates how a CALL, DXFR, or XFER is to be implemented. For the CICS target environment, the linkage table is also used to indicate whether a VSAM or transient data queue is to be accessed locally or remotely. For example, the linkage table might specify that a CALL to application XYZ in the MVS CICS target environment is to be implemented as a remote call passing data in the CICS COMMAREA.

For Cross System Product prior to 4.1, there was no linkage table. For Cross System Product 4.1 and releases of VisualAge Generator prior to 3.0, the linkage table was in a separate file.

- Resource association file that indicates for a specific file how it is to be implemented in a specific target environment. For example, a serial file in the MVS CICS target environment might be implemented as a VSAM file, as a transient data queue, as a temporary storage queue, or as a CICS spool file.

For Cross System Product, resource association information was stored in the MSL. For releases of VisualAge Generator prior to 3.0, resource association information was stored in a resource association file outside the MSL.

- Bind control commands that provide information needed for binding the DB2 application plan on an MVS system.

For Cross System Product/AE, bind control information was not used. Cross System Product COBOL generation used bind control commands and

these were stored in separate files outside the MSL. For releases of VisualAge Generator prior to 3.0, bind control commands were stored in separate files.

- Linkage editor control statements that provide linkage editor information for MVS, VSE, and VM systems.

For Cross System Product/AE, linkage editor control statements were not used. Cross System Product COBOL generation used linkage editor control statements and these were stored in separate files outside the MSL. For releases of VisualAge Generator prior to 3.0, linkage editor control statements were stored in separate files.

With VisualAge Generator 3.0 and later, generation options files, linkage tables, resource association files, bind control commands, and linkage editor control statement files are methods within VAGen part classes in ENVY. Thus all the data required for test and generation is contained in a single library management system.

Table 8 shows the correspondence between the types of control information and VAGen part classes.

Table 8. Control Information and VAGen Part Classes

Control Information	VAGen Part Class
Generation Options	VAGenOptions
Linkage Table	VAGenLinkages
Resource Associations	VAGenResources
Bind Control Commands	VAGenBindControls
Linkage Editor Control Statements	VAGenLinkEdits

MSL concatenation

In Cross System Product and releases of VisualAge Generator prior to 3.0, MSLs could be concatenated. Specifying the MSL concatenation sequence was the way in which you specified where to look for the members needed by your applications. When MSLs were concatenated, for test and generation, only the first found member with a given name was used. For viewing, members from MSLs other than where the first found member was located could be referenced. If changes were made to a member in the MSL concatenation sequence, the changed member was stored in the read/write MSL (first MSL in the concatenation sequence).

With VisualAge Generator 4.0, there is no concept similar to the first found member in an MSL concatenation. All editions are available in the ENVY repository. However, only one edition of a part can be loaded into your workspace at a time. Browsers are available to compare editions within the

ENVY repository before loading them into your workspace to determine which one is the required level of code.

You can use projects to group the code for a particular level. For example, you might have a project for production that indicates the version of each package that is in production. You can also specify a Project List Part that includes a list of projects to be loaded with a particular project.

Specifying a Project List Part for a project is similar to specifying the MSL concatenation sequence. Project List Parts provide a way for you to ensure that all the parts needed to run a particular group of programs are loaded into your workspace together. Project List Parts must be defined in such a way that only one edition of a part is loaded into your workspace. Therefore, PLPs resolve the problem of first found parts. For more information on Project List Parts, see:

- “Project List Part” on page 57
- “Creating a Project List Part (PLP)” on page 154

Functional organization

In Cross System Product and releases of VisualAge Generator prior to 3.0, members were grouped together into MSLs. Generally, an MSL contained all the members for a particular subsystem. Because the MSL was the only method for grouping members by function, the functions tended to be large. In Cross System Product the number of MSLs in a concatenation sequence was limited to 6. This also contributed to having a large number of members in each MSL.

With VisualAge Generator 4.0, the project and the ENVY package provide a two-level capability for grouping parts. The project is the higher level of organization and more closely resembles an MSL in terms of the number of parts. ENVY packages enable you to organize your parts into smaller groups than was reasonable to do with MSLs. This provides more capabilities in terms of controlling access to the parts, finding a part, and limiting the number of parts displayed in the VAGen Parts Browser window.

Member associations

For Cross System Product and releases of VisualAge Generator prior to 3.0, members could have associates — other members that were referenced within the member. With VisualAge Generator 4.0, the same associations between 4GL VAGen parts still exist. For example:

Member Type	Member Types of Possible Associates
Data Item	None
Process	None
Statement Group	None

Record	Any global data items
Table	Any global data items
PSB	Records for the segment records, any global data items used by those records, and any item named as a secondary index field in the PSB
Map	Any other maps in the same map group, any statement group or table used as an edit routine, and any global data item in the table
Map Group	Any maps in the map group and their associates
Application	Map groups, maps, records, tables, PSB, processes, statement groups, and their associates. Any global data item referenced directly or indirectly was included. For example, a data item used as a called parameter was included as an associate of the application. Only maps that were actually used by the application were included in the associates.

Notes:

1. References to processes and statement groups in the above list represent the MSL terminology. These VAGen part types have been merged into the new VAGen function part type.
2. References to global data items in the above list represent the MSL terminology. Global data items have been renamed to *shared data items* in VisualAge Generator 4.0.
3. References to applications in the above list represent the MSL terminology. Applications have been renamed to *programs* in VisualAge Generator 3.0 and later.

For a specific member (for example, application XYZ), the only associations that could be detected were those that existed in the current MSL concatenation sequence, using the first found members.

For VisualAge Generator 4.0, the only associations that can be detected are the ones that exist within the current workspace.

For releases of VisualAge Generator prior to 3.0, the associates of GUIs were records, tables, processes, statement groups, or other embedded GUIs, and their associates. An external GUI and its associates were not included.

With VisualAge Generator 4.0, for a view, the following are associates:

- 4GL parts — records, tables, and functions and their associates.
- For any embedded view, its associates. However, the embedded view itself is not included as an associate.

Chapter 8. General migration considerations for VAGen 2.x and Cross System Product to Java

Consider the following if you are migrating from VisualAge Generator 2.x or Cross System Product to VisualAge Generator 4.0 on Java:

- “Migration paths”
- “Automatic conversions during 4.0 migration” on page 48
- “Migrating GUIs” on page 49
- “Resolving duplicate member names” on page 49
- “Establishing naming conventions” on page 50
- “Organizing your code for ENVY” on page 52
- “Assigning ownership” on page 53
- “Storing control information” on page 54
- “Dual maintenance” on page 57
- “Migrating from Cross System Product” on page 57
- “Migrating from OS/2 to Windows NT” on page 58

Also see “Chapter 9. Pre-migration checklist” on page 59 before you begin to migrate code to VisualAge Generator 4.0.

In addition, if you are migrating from Cross System Product, it is strongly recommended that you read *Migrating Cross System Product Applications to VisualAge Generator* and “Appendix D. Notes on Cross System Product migrations” on page 395 before you migrate.

Migration paths

Depending on your current platform and whether you want to migrate to Smalltalk, Java, or both, different migration paths are available and different considerations apply.

Table 9 gives a brief overview of the migration options available for the Java platform. See the referenced chapters for step-by-step procedures for each migration option.

Table 9. Migration Options

Migrating from	Tools to Use	Details on Using the Tools
VisualAge Generator 2.x (GUI)	No tool	Recreate parts on Java platform

Table 9. Migration Options (continued)

Migrating from	Tools to Use	Details on Using the Tools
VisualAge Generator 2.x or Cross System Product (non-GUI)	<ul style="list-style-type: none"> • MSL Migration Assistance Tool • VAGen Import 	<ul style="list-style-type: none"> • “Chapter 10. The MSL Migration Assistance Tool on Java” on page 61 • “Chapter 13. Running the MSL Migration Assistance Tool on Java” on page 115 • “Chapter 14. Using VAGen Import to migrate VAGen 2.x and Cross System Product non-GUI code to Java” on page 149

Automatic conversions during 4.0 migration

VisualAge Generator 4.0 automatically makes the following changes to your applications during migration:

- Members become VAGen parts.
- Process and statement group members are converted to function parts.
- References to processes and statement groups are converted to the new syntax requirements for functions with no parameters.
- PERFORM statements and Unconditional Branch statements are no longer supported and are migrated to Function Invocation statements.
- Subscript parentheses are changed to brackets in VisualAge Generator item names in the following places:
 - 4GL statements in functions (processes and statement groups)
 - Host variable names in SQL statements
 - Comparison value item in DL/I specifications
 - EZEDLPCB is used in a called parameter list
- Calls to EZE service routines are converted to the corresponding function invocation statement. A statement to set the value of EZEREPLY is also added before the function invocation.

In addition, the names of part classes, control information files, and VisualAge Generator palette parts are changed during migration to VisualAge Generator 4.0:

- Table 20 on page 361 shows the conversions made between member types and VAGen 4.0 part classes during 4.0 migration.

- Table 21 on page 361 shows the correspondence between control information files and VAGen 4.0 part classes.
- Table 22 on page 362 shows the changes made to VisualAge Generator palette parts names during 4.0 migration.
- Table 23 on page 362 shows the changes made to VAGen Templates part class names and repartition during 4.0 migration.

Migrating GUIs

VisualAge Generator 4.0 does not provide a way to migrate existing GUI parts to the Java platform. You need to recreate your GUI parts for Java.

When you are migrating MSLs that contain both GUIs and 4GL parts, only the 4GL parts can be migrated to VisualAge Generator 4.0 on Java using the MSL Migration Assistance Tool. After you migrate the 4GL parts, recreate the GUI parts in VisualAge Generator 4.0 on Java and add them to the migrated package containing the corresponding 4GL parts.

For example, suppose MSL1 has the following parts:

```
App1
Process1
Record1
Gui1
Gui2
```

When you use the MSL Migration Assistance Tool to migrate App1 into pkg1 in VisualAge Generator 4.0 on Java, Process1 is migrated to Function1 and Record1 is migrated to Record1. You must create (rewrite) new GUI parts in pkg1 to replace Gui1 and Gui2 from the MSL.

Resolving duplicate member names

Duplicate member names should be resolved (or at least understood) before starting the migration. Duplicates can arise in the following situations:

- A duplicate member was accidentally copied into the wrong MSL and was never deleted. In this case, you must determine which is the correct version of the member.
- Duplicate members that have the same name, but are different member types must be resolved. This might require renaming one of the members and changing all references to it.
- The duplicate member is in a staging MSL or test MSL and represents work-in-progress. In this case, the changed member should be loaded after the production MSL(s) are loaded and versioned within ENVY. Alternatively, these duplicate members might be ones that were created, then forgotten, and never intended for production.

- The duplicate member is due to a demonstration that needs to reflect the state of the demonstration at different times (for example, the initial state of the MSL, the MSL after some changes have been made, and the MSL after another set of changes have been made). In this case, it might be better to create a complete MSL representing the entire set of code at various times during the demonstration and make a separate package for each complete MSL.

Establishing naming conventions

You probably already have naming conventions for parts like programs, processes, records, and so on that you are migrating from Cross System Product or VisualAge Generator 2.x. You can continue to use your existing naming conventions, because VisualAge Generator 4.0 retains existing part names during migrations from Cross System Product and VisualAge Generator 2.x.

However, VisualAge Generator does change some existing part *types*, as well as making some syntax conversions. For example, process members and statement group members are both changed to the function part type. Therefore, you might want to establish a naming convention for function parts that is similar to your conventions for processes and statement groups.

See “Automatic conversions during 4.0 migration” on page 48 for a list of changes that VisualAge Generator makes to your existing code during migration.

You need to establish naming conventions for the new package and project objects that you must create in Java for your migrated code. Your naming conventions should include the following:

- A unique prefix (possibly three characters) so that all of your packages will be grouped together in the VisualAge on Java workspace. For example, you might choose to prefix all your packages with a company prefix of *xyz*.

If more than one grouping of packages exists within a single project, each group of packages could have a different secondary prefix. You can use Java dot notation to include your global prefix as well as the secondary prefix for each group.

For example, if you have an Accounting system and a Payroll system, you could use your company prefix of *xyz* and then add a secondary prefix of *acct* for the Accounting system and *pay* for the Payroll system. When your packages are all loaded into your VisualAge workspace, they would appear in this order:

```
xyz.acct.pkg1
xyz.acct.pkg2
xyz.pay.pkg1
xyz.pay.pkg2
```

As an alternative, you could leave off the company prefix and create first-level prefixes that describe the tasks performed by the packages. Using prefixes that start with different letters can be an advantage, because some VisualAge windows allow you to enter one letter to quickly tab to the group of packages that start with that letter.

- Try to avoid having the same package appear in multiple projects, because projects containing duplicate packages cannot be loaded together.
- A naming convention for your projects. For example, if you decide to use different projects for development, system test, acceptance test, and production, you might use the following as the projects names:

```
Project Name:          xxxxxxxxDevTestProject
                    xxxxxxxxSysTestProject
                    xxxxxxxxAccTestProject
                    xxxxxxxxProdProject

Version Names:         could just be sequential
```

In this naming scheme, `xxxxxxx` is the subsystem name.

If you decide to use a single project and use a build approach in which you build (generate programs) once a week, you might use the following:

```
Project Name:  xxxxxxxxProject

Version Names  rrr nn.p
```

In this naming scheme:

<code>xxxxxxx</code>	Is the subsystem name.
<code>rrr</code>	Is the release name.
<code>nn</code>	Is the build within the release.
<code>p</code>	Is a point between builds.

Use the same version naming scheme for the packages and classes. The `p` allows the class owners and package managers to create as many versions as they need to between builds. The `rrr nn` means that the classes, packages, and projects changed during the same week all carry the same release and version number. This helps everyone remember when changes were made.

- Unique **version names** for your migrated packages and projects. Using the default naming convention (1.0, 1.1, and so on) might be the best way to start.

In addition to your company-unique naming conventions, there are some general Java naming conventions that you need to follow:

- Java expects package names to be in lowercase characters.

- Java uses “dot notation” for package names, where each word in the name is separated by a dot (.).

Organizing your code for ENVY

For an overview of the differences between MSLs and ENVY, see “Chapter 7. Comparing MSLs and library management on VAGen 4.0 on Java” on page 39.

Note: Performance is best when you limit a package to a maximum of 600-700 VAGen parts per VAGen part type. It is also recommended that you use no more than 50 classes per package. When considering this maximum, remember that processes and statement groups are both collapsed into the VAGenFunctions class.

Functional organization

ENVY works best when packages are organized along functional lines. This provides some benefits:

- Helps limit the scope of a change to one or a few packages.
- Supports the concept of ownership described in “Assigning ownership” on page 53.

You should divide your MSLs into functional areas, with one package for each functional area. Some examples are:

- A DBA MSL that contains data items and record definitions. This might be small enough to become a single package. However, you might also decide to organize it into smaller packages by dividing it using one or more of the following suggestions:
 - One package for the tables and data items that are shared by multiple subsystems and a separate package for each subsystem that contains data items and records that are unique to that subsystem.
 - One package for the tables and data items used in a particular target environment.
 - One package for a group of closely related tables, where you want smaller packages than any of the previously mentioned techniques.
- An MSL containing common code might be small enough (if you only have a few common parts) to become a single package. However, you should consider organizing it along functional lines such as:
 - Error and message handling
 - Security support
 - Audit log or journal support
 - Tools only used during development, not in the production system

You can also use the following techniques to assist you in organizing your parts into functional areas:

- You might have used a naming convention to help you identify parts of a system. For example, PAYWK might be the prefix for all the parts that are unique to processing the weekly payroll. PAYSL might be the prefix for all the parts that are unique to the salary payroll system. In this case, you could put all the parts that start with PAYWK into one package (named *pay.weekly*) and the parts that start with PAYSL into a different package (named *pay.salary*). Then combine these two packages into one payroll project.
- For a menu system, if there are 10 items on the main menu, you might start by putting all programs that are used when the user selects option 1 into one package, all programs that are used when the user selects option 2 into another package, and so on. If you use this technique, consider called programs as well as programs that are transferred to using XFER or DXFR. Then combine all the menu-item packages into one project for this menu system.

This technique has the advantage of including programs that pass data among themselves into the same package. If you have multiple layers of menus, you could use this technique at the lowest level of menus.

- You might have designed and organized your applications in the past using a technique that is unique to your organization. Using the same organizing scheme when you migrate to Java would provide continuity for your development organization and reduce the time needed for skills transfer to the new development environment.

With any of these techniques, if you use the MSL Migration Assistance Tool, it can help you identify common code that will be shared by several packages and help you split this common code into separate packages and projects.

Assigning ownership

Each class within a package has an owner. This includes visual parts and VAGen part classes. Each package and each project has an owner. The classes within a package can have different owners, and the owners can be different from the owner of the package and project containing those classes.

Your ownership strategy should reflect how development and maintenance responsibilities are divided in your company. You need to provide a backup mechanism in case an owner is unavailable.

Keep the following in mind when assigning ownership:

- In general, parts should be grouped into functional components so that ownership can be assigned for maintenance.
- If one developer typically handles an entire program, then that developer should be the owner for the corresponding package and all the classes in that package.

- Make the packages small enough that a single developer does not own the entire system (unless the single developer is the only person who ever works on that system).
- Make the packages small enough that very few people are working in the same package at the same time.

Storing control information

In VisualAge Generator 2.2, files were used to store the following control information:

- Linkage table
- Resource association file
- Generation options
- Bind commands
- Linkage editor control statements

In VisualAge Generator 4.0, the control information is stored in parts in the ENVY repository.

You probably need to use different generation options for development, system test, acceptance test, and production. For example, you might need to specify different load libraries for the outputs of preparation or point to different DB2 systems for different levels of testing. Similarly, you might need different resource associations, linkage editor control statements, and bind statements.

There are two alternative techniques for organizing control information:

- Use multiple control packages.
- Use a single control package.

Multiple control packages

One option is to store the control information for each level of test in a different package. The package names could use the naming convention:

```
sss.control.common  
sss.control.dev  
sss.control.sys.test  
sss.control.acc.test  
sss.control.prod  
sss.control.emergency
```

where:

- | | |
|---------------|--|
| sss | Is the subsystem ID that is the same prefix you use for the other packages in the project. |
| common | Is used for control information that is common to test and |

production. None of the part names in Common is duplicated in dev, sys.test, acc.test, prod, or emergency.

dev	Is for the developers to use.
sys.test	Is for systems test.
acc.test	Is for acceptance test.
prod	Is for production.
emergency	Is for emergency fixes.

The parts might be named as follows:

```
sssMvsCicsGenOpts
sssVseCicsGenOpts

sssResourceAssociations

sssLinkageTable

pppppp.BDC (for bind information that is unique to a program)
pppppp.LED (for linkage editor information that is unique to a program)
```

With this technique, you have `sss.control.common` and only one of the other `sss.control.xxxx` projects loaded into your workspace. During test and generation, the part name you specify for a particular type of control information is the same part name regardless of whether you are working at the development, system test, acceptance test, or production level of code.

The projects for development, system test, acceptance test, and production each include the `sss.control.common` and `sss.control.xxxx` packages that correspond to their level of testing.

`sss.control.emergency` is a special control package that is very similar to `sss.control.production`. However, it specifies the load libraries needed for emergency fixes rather than the normal production load libraries.

Single control package

A second option is to collect all the control information into a single package. This can make it easier to maintain the control information. The parts might be named as follows:

```
sssCommonMvsCicsGenOpts
sssXxxMvsCicsGenOpts

sssCommonVseCicsGenOpts
sssXxxVseCicsGenOpts

sssXxxMvsResourceAssociations
```

sssXxxVseResourceAssociations

sssXxxMvsLinkageTable

sssXxxVseLinkageTable

ppppppp.BDC (for bind information that is unique to a program)

ppppppp.LED (for linkage editor information that is unique to a program)

In this naming scheme:

sss Is the subsystem name.

Common

Is used for control information that is common to test and production.

Xxx Is the level of test or production (for example, dev, sys.test, acc.test, prod, or emergency).

ppppppp

Is the name of a program.

Generation options

/DESTNAME, /COBOL, and /NLS are gone.

/BIND and /LINKEDIT have changed. They do not point to a directory, just to a suffix. For example, if your naming convention for linkage editor control parts is ppppppp.LED, where ppppppp is the program name, you should set /LINKEDIT=LED for the generation option.

/OPTIONS, /LINKAGE, and /RESOURCE have also changed. They do not point to a directory, just to a part. For example, you might set /OPTIONS=sssMvsCicsGenOpts.

/PROJECT generation option

A new /PROJECT option for the GENERATE command is available for VisualAge Generator 4.0 on Java. Using /PROJECT, you can generate C++ and COBOL from VisualAge Generator parts stored in the Java repository. Multiple /PROJECT options can be specified on the same GENERATE command. For example, this command causes both My Project and Your Project to be loaded:

```
hptcmd generate MYPROG /PROJECT="My Project","Version 1.0"  
/PROJECT="Your Project","Version 29.5" ...
```

Note: You cannot use /PROJECT for two projects specified on the same GENERATE command that contain packages with the same name.

See the *VisualAge Generator Generation Guide* for more information about the /PROJECT option.

Project List Part

VisualAge Generator on Java also offers a new Project List Part (PLP). The PLP is a generation options part in which you specify a list of projects to be loaded together. You can specify the list of projects once in the PLP part and maintain the list in only this one place. The PLP eliminates the need to specify all related projects each time the projects are loaded. You can create a PLP for each group of projects that you want to load together.

See “Creating a Project List Part (PLP)” on page 154 for information on how to create project list parts.

Dual maintenance

The external source format file for 4GL parts that you export from VisualAge Generator 4.0 is not compatible with the external source format file for VisualAge Generator 2.2 or Cross System Product. Therefore, if you migrate a subsystem that shares common parts with a subsystem that you will migrate at a later time, you have the following alternatives for maintenance of the common parts:

1. Maintain the common parts on VisualAge Generator 2.2 or earlier using MSLs, and when you are satisfied with the changes:
 - a. Export an external source format file from the MSL for the changed parts.
 - b. Import the external source format file into ENVY using the **Defined package** radio button so the changes will go into the same ENVY package in which the parts are already located.
2. Make the same changes to both the part in ENVY using VisualAge Generator 4.0 and the corresponding MSL member using VisualAge Generator 2.2.

Migrating from Cross System Product

When you migrate from Cross System Product, you have the following additional considerations:

- If you are migrating from Cross System Product 3.3 or earlier, you are migrating from interpretive CSP/AE to generated COBOL or C++. Refer to *Migrating Cross System Product Applications to VisualAge Generator* (SH23-0244-01) for information on the compatibility considerations involved in this portion of the migration.
- You must also plan for how to handle the workstation environment:
 - Backup and recovery
 - Whether data will reside on the workstation or on the host
 - How to handle calls to non-VisualAge Generator packages when you are using the test facility

- How to handle functions that were only supported when running the test facility in the CICS environment (for example, CREATX and the use of transient data queues).
- Early releases of Cross System Product allowed invalid data to be stored in the MSLs. If an external source format file contains an invalid member, you must either correct the member on Cross System Product and export the external source format file for the member again or correct the external source format file by editing it. The following are examples of the types of problems that you might encounter:
 - A data item that has a length of 0
 - A map that contains a NUM field with a date edit, but the field is not long enough to contain a date
 - Members in the MSL that contain generated Cross System Product code that really should have been stored in an ALF
- VisualAge Generator does not allow generation of packages that check a non-SQL data item for the NULL state. VisualAge Generator rejects as invalid any conditional statement (IF, TEST, WHILE) that tests the state of a non-SQL data item for the NULL condition. However, this type of statement was allowed in CSP 3.3. Therefore, you should rewrite any statements that check non-SQL data items for the NULL state before migrating CSP-generated applications that contain them.

Migrating from OS/2 to Windows NT

If you are changing from the OS/2 to the Windows NT development platform, be sure to review “Changing from OS/2 to Windows NT” on page 119 before you start to migrate.

Using the migration log file

The migration log file is named **mslmig.log** and is located in the same directory as your workspace. This log file stores the results of **Write to File** operations. You can use a text editor to view and print the log file.

Chapter 9. Pre-migration checklist

1. Before you migrate, you should first read the following:
 - “Chapter 7. Comparing MSLs and library management on VAGen 4.0 on Java” on page 39
 - “Chapter 8. General migration considerations for VAGen 2.x and Cross System Product to Java” on page 47
 - “Chapter 10. The MSL Migration Assistance Tool on Java” on page 61
 - “Chapter 11. Migrating production and work-in-progress MSLs to VAGen 4.0 on Java” on page 75
 - “Chapter 12. VAGen on Java case studies based on various MSL structures” on page 85
 - VisualAge Generator 4.0 readme file
2. Save a clean copy of your VisualAge for Java workspace. This clean copy should not contain any of your application code. Copy *ide.icx* to *ideclean.icx*, and store the clean copy on your LAN so that all developers have access to it. Saving copies of the *hpt.ini*, *ide.ini* and *ivj.dat* files is also recommended.
3. Check with IBM support to see if there are any fixes available for the VisualAge Generator 4.0 MSL Migration Assistance Tool. If there are, import the fixes and then load them into your 4.0 image.
4. For better performance during migration, you might want to do the following:
 - Dedicate a single workstation or server to use for migration. Attempting to use multiple workstations limits VisualAge Generator’s ability to detect duplicates and common code and to determine when previously missing parts have been found. Having the product provide this information more than compensates for any potential time savings that might be gained by using several workstations.
 - Copy your MSLs to the workstation you set up to run the VisualAge Generator migration tools, or make sure your code libraries are on a server that your migration workstation can access. This improves performance for the migration tools.
 - If you are migrating from VisualAge Generator 2.x or Cross System Product, you will probably use the MSL Migration Assistance Tool to migrate to VisualAge Generator 4.0. Run the tool on the server where the ENVY repository is located. This improves performance for committing parts to ENVY.

If you plan to use the MSL Migration Assistance Tool, review “Collecting your source code” on page 117 and “Handling code page changes” on page 118 for information on how to make your code from

Cross System Product or previous versions of VisualAge Generator available to the MSL Migration Assistance Tool.

5. Contact your local IBM representative to learn more about VisualAge Generator service offerings that can help you with migration.

Chapter 10. The MSL Migration Assistance Tool on Java

The MSL Migration Assistance Tool is designed to assist in migrating MSLs to the ENVY library manager.

Note: Although the MSL Migration Assistance Tool can help determine when members are not found, it cannot locate missing members. Similarly, the MSL Migration Assistance Tool can help determine when duplicates of a given member exist, but it cannot make the determination as to which is the correct or current level of the member.

The migration process works from an MSL directory structure. In some cases, you will not have an MSL on the workstation. For example, if you are migrating from Cross System Product, your MSLs are VSAM files on the host. The MSL Migration Assistance Tool enables you to create MSL directories from external source format files. You can then use these MSL directories during your migration. You do not need to install VisualAge Generator 2.2 to create the MSL directories to use during migration.

The MSL Migration Assistance Tool enables you to select parts (members) from an MSL or MSL concatenation, group them into a package or series of packages, and move them to a “sandbox”. When you move a part to the “sandbox”, its associates also move. The packages in the sandbox are in the workspace, but are not in the ENVY repository yet. This allows you to manipulate the packages and to rearrange the VAGen parts within the packages until you are satisfied with the organizational structure. Only one version of a part can be in the sandbox at a time.

After you are satisfied with your organizational structure for the packages, you can commit the packages to the ENVY repository. Committing the packages creates the following:

- An edition of the project in the ENVY repository
- An edition of the packages in the ENVY repository
- Any needed VAGen part classes for the 4GL member types
- The VAGen parts for the 4GL members

After the packages are in ENVY, you can use any of the ENVY library management functions such as:

- Versioning and releasing the VAGen part classes and parts
- Versioning and releasing the packages
- Creating Project List Parts (PLPs)
- Versioning projects

- Assigning a class owner or a package owner

Overview of using the MSL Migration Assistance Tool

You can use these steps to start the MSL Migration Assistance Tool:

1. From the IBM VisualAge Generator 4.0 folder, select **VAGen Developer 4.0 on Java with Migration**.
2. From the VisualAge for Java Workbench window, select **Workspace->Open VAGen Parts Browser**. The VAGen Parts Browser window is displayed.
3. From the VAGen Parts Browser window, select **Tools->Migration->MSL Migration**. The main MSL Migration Assistance Tool window, called MSL Migration Part List, is displayed, as shown in Figure 4.

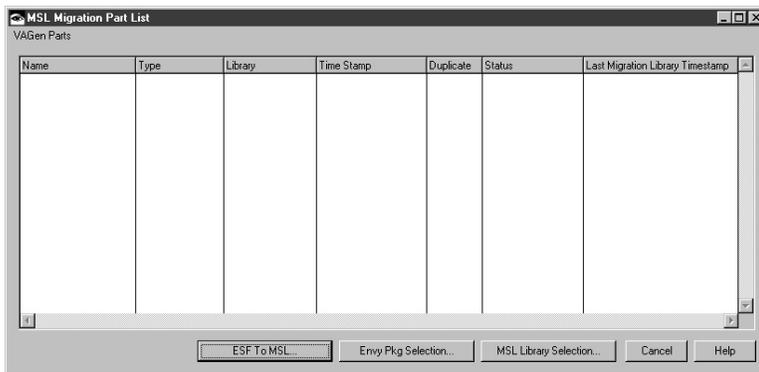


Figure 4. MSL Migration Assistance Tool window

Note: You must start VisualAge Generator 4.0 on Java using the **VAGen Developer 4.0 on Java with Migration** option for the migration tools to be loaded in your workspace. After you have finished migrating all your code, you can start VisualAge Generator 4.0 without the migration tools to save memory and increase performance.

You can also start VisualAge Generator 4.0 with the MSL Migration Assistance Tool from a command line, using these steps:

1. Open a command line window to the directory where the `ide.icx` file is stored, and enter the command `ide /vgmig`. (Note the blank space after `ide`.)
2. From the VisualAge for Java Workbench window, select **Workspace->Open VAGen Parts Browser**. The VAGen Parts Browser window is displayed.
3. From the VAGen Parts Browser window, select **Tools->Migration->MSL Migration**. The main MSL Migration Assistance Tool window, called MSL Migration Part List, is displayed.

The sections that follow provide an overview of how to use the MSL Migration Assistance Tool and the purpose of each window. Reading these sections will help you understand the following chapters that describe how you might organize your members into ENVY packages:

- “Chapter 8. General migration considerations for VAGen 2.x and Cross System Product to Java” on page 47
- “Chapter 11. Migrating production and work-in-progress MSLs to VAGen 4.0 on Java” on page 75
- “Chapter 12. VAGen on Java case studies based on various MSL structures” on page 85

After you have read the chapters on organizing your members into ENVY packages, you will be ready to use the more detailed procedures provided in “Chapter 13. Running the MSL Migration Assistance Tool on Java” on page 115.

Building MSL directories

Before you can begin the migration process, you must have MSLs to migrate. One of the following situations will apply:

- You are migrating from Cross System Product, so you have never had MSLs on the workstation. Create one external source format file for each host MSL and download the external source format files to the workstation.
- You are migrating from VisualAge Generator but have used TeamConnection and do not have MSLs. You might want to create one external source format file for each component to help preserve the organizational structure you created in TeamConnection.
- You are migrating from VisualAge Generator and are changing from an OS/2 development environment to a Windows NT development environment. If you have not used special characters, you might be able to use the OS/2 MSLs. However, some special characters are at different code points in Windows NT and OS/2. If you have special characters other than the not sign (¬), you cannot use the OS/2 MSLs due to code page differences between the two environments. Create one external source format file for each MSL. See “Changing from OS/2 to Windows NT” on page 119 for information on converting between code pages using these external source format files before you create MSLs on Windows NT.

After you have created your external source format files, from the MSL Migration Part List window, select the **ESF to MSL** push button. The MSL Migration Assistance Tool prompts you for the name of an external source format file, as shown in Figure 5 on page 64.

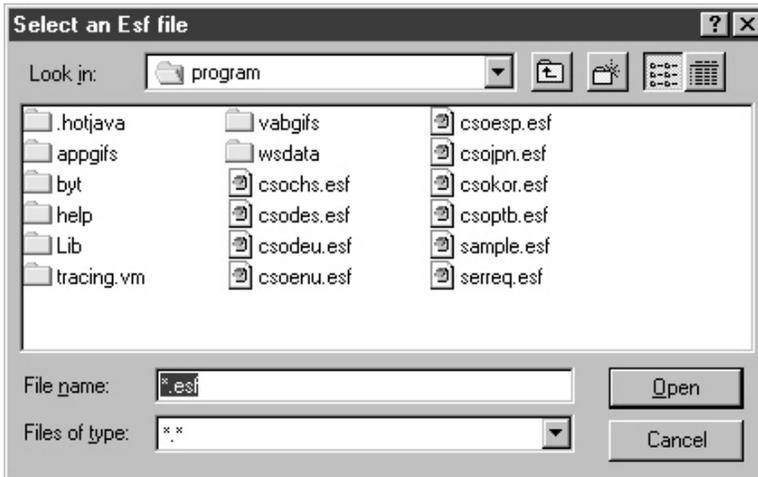


Figure 5. Prompting for an external source format file name

The MSL Migration Assistance Tool then prompts you for the name of a directory into which the MSL members are to be placed, as shown in Figure 6.

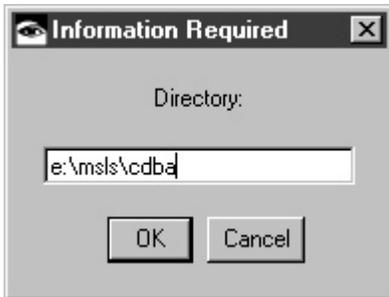


Figure 6. Prompting for an MSL directory name

After you have specified this information, the MSL Migration Assistance Tool creates the MSL members from the external source format file. If you specify a directory that does not exist, the MSL Migration Assistance Tool creates it for you.

Note: The VisualAge Generator 4.0 automatic conversions occur at this time. Therefore, processes and statement groups will become functions.

Selecting MSLs - using the MSL Library Selection window

Before you can begin moving parts to the sandbox, you first need to specify the MSLs that you want to process. From the MSL Migration Part List

window, select the **MSL Library Selection** push button. The MSL Library Selection window is displayed, as shown in Figure 7.

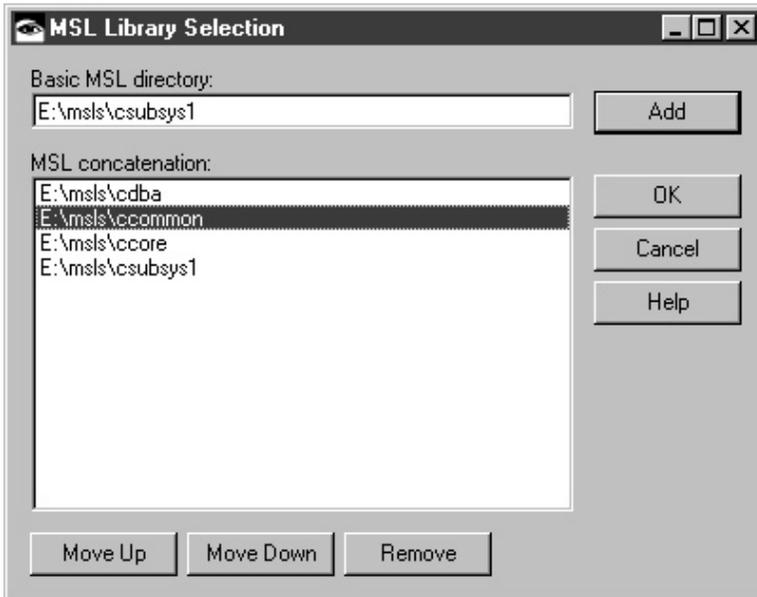


Figure 7. MSL Library Selection window — Adding an MSL to the concatenation

From the MSL Library Selection window, you can specify the drive and path for one or more MSL directories. Specify the drive and path for one MSL in the *Basic MSL directory* field and then select the **Add** push button. Repeat this until you have specified the information for all the basic MSLs in the concatenation sequence that you want to process. You can use the **Move Up**, **Move Down**, and **Remove** push buttons to change the concatenation sequence.

After you specify the MSLs and the MSL concatenation sequence, select the **OK** push button. The Part List Selection Criteria View window is displayed and lists the MSL directories.

Selecting parts for a part list - using the Part List Selection Criteria View window

The Part List Selection Criteria View window is similar to a Member Selection List in Cross System Product or VisualAge Generator.

The Part List Selection Criteria View window is shown in Figure 8 on page 66.

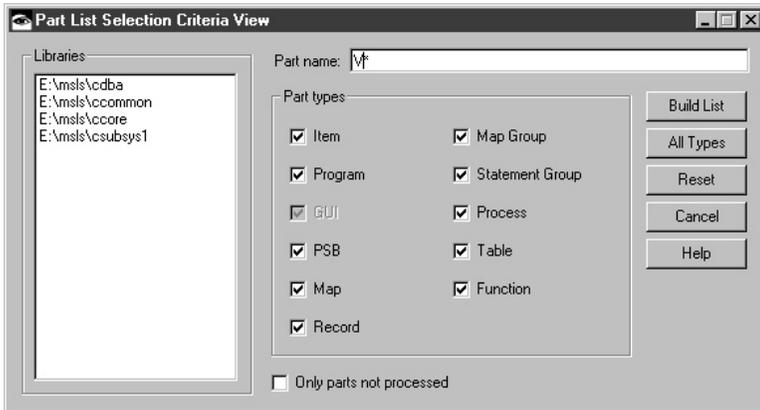


Figure 8. Part List Selection Criteria View window

From the Part List Selection Criteria View window, you can do the following:

- Specify a portion of a part name using a wildcard.
- Specify which part types you want to appear on the MSL Migration Part List window.

However, the Part List Selection Criteria View window differs from a Member Selection List in the following ways:

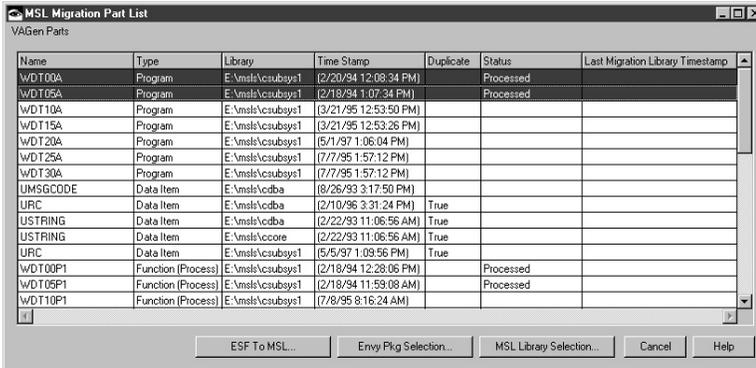
- It allows you to specify whether you want to see all parts that satisfy the part type criteria or whether you only want to see those parts that satisfy the part type criteria that have not yet been moved to the sandbox. This is controlled by the **Only parts not processed** toggle button.
- It does not allow you to specify which MSLs from the concatenation sequence are to be used. All listed MSLs are always used. If duplicate members exist, they are all shown.
- **Function** is listed as a part type. If you are migrating using external source format files to create pseudo MSLs, select the function part type. If you are migrating using your VisualAge Generator 2.x MSLs, select the **Process** and **Statement Group** part types.

Note: The check box for the GUI part type is disabled in Figure 8 because GUIs cannot be migrated to VisualAge Generator 4.0 on Java.

When you select the **Build List** push button, the parts that satisfy the selection criteria appear in the MSL Migration Part List window.

Selecting parts to move to ENVY - using the MSL Migration Part List window

The MSL Migration Part List window lists the parts that satisfy the selection criteria specified in the Part List Selection Criteria View window.



Name	Type	Library	Time Stamp	Duplicate	Status	Last Migration Library Timestamp
WDT00A	Program	E:\msl\vcsubsys1	{2/20/94 12:08:34 PM}		Processed	
WDT05A	Program	E:\msl\vcsubsys1	{2/19/94 1:07:34 PM}		Processed	
WDT10A	Program	E:\msl\vcsubsys1	{3/21/95 12:53:50 PM}			
WDT15A	Program	E:\msl\vcsubsys1	{3/21/95 12:53:26 PM}			
WDT20A	Program	E:\msl\vcsubsys1	{5/1/97 1:06:04 PM}			
WDT25A	Program	E:\msl\vcsubsys1	{7/7/95 1:57:12 PM}			
WDT30A	Program	E:\msl\vcsubsys1	{7/7/95 1:57:12 PM}			
UMSGCODE	Data Item	E:\msl\vcdba	{8/26/93 3:17:50 PM}			
URC	Data Item	E:\msl\vcdba	{2/10/96 3:31:24 PM}	True		
USTRING	Data Item	E:\msl\vcdba	{2/22/93 11:06:56 AM}	True		
USTRING	Data Item	E:\msl\vcdbcore	{2/22/93 11:06:56 AM}	True		
URC	Data Item	E:\msl\vcsubsys1	{5/5/97 1:09:56 PM}	True		
WDT00P1	Function (Process)	E:\msl\vcsubsys1	{2/18/94 12:28:06 PM}		Processed	
WDT05P1	Function (Process)	E:\msl\vcsubsys1	{2/18/94 11:59:08 AM}		Processed	
WDT10P1	Function (Process)	E:\msl\vcsubsys1	{7/8/95 8:16:24 AM}			

Figure 9. MSL Migration Part List window with parts

From the MSL Migration Part List window, you can select the parts that are to be moved to the sandbox. The MSL Migration Assistance Tool moves the selected parts **with their associates** to the sandbox. You can specify that the selected parts are to be moved in the following ways:

- Into a single package
- Into multiple packages, with each selected program becoming a different package
- Into a package that is already in the sandbox

When you move parts to the sandbox, it is possible that an associate of the part you are currently moving is already in the sandbox, but in a different package from the one that you have specified as the target for this move.

In this case, the associates that are shared by more than one package are automatically moved into a separate package called `package.noden`, where `n` is a number to distinguish different package nodes. Each `package.node` represents a group of parts that have the same set of required packages.

When you move one or more parts to the sandbox, the VG Part Prerequisites View window is displayed and shows the current state of the sandbox.

For the purposes of migration, the associates of a part are the same as in Cross System Product or VisualAge Generator 2.x or earlier.

Note: Process parts and statement group parts are displayed in the MSL Migration Part List window as *function* parts. If you used the **ESF to MSL** push button to create an MSL that you could migrate, all your process and statement group parts were converted to function parts when the MSL was created. These parts are displayed in the MSL Migration Part List window with a type of **function**. The MSL Migration Assistance Tool is unable to discern the origin of these function parts. If, however, you used the **MSL Library Selection** push button to select an existing MSL for migration, your process and statement group parts are displayed in the MSL Migration Part List window with a type of **function (process)** or **function (statement group)**. This allows you to determine which function parts were originally process parts and which were originally statement group parts.

After you think you have moved all the parts to the sandbox, you can verify this from the Part List Selection Criteria View window by selecting the **Only parts not processed** toggle button, then selecting the **All Types** push button, and then selecting the **Build List** push button. If everything has been moved to the sandbox, the MSL Migration Part List window will not list any parts.

Special columns in the MSL Migration Part List window

The MSL Migration Part List window provides information to help in determining which parts to move to the sandbox. The **Status**, **Duplicate**, and **Last Migration Library Timestamp** work together to help you resolve duplicate or missing parts.

The **Status** column indicates the following:

Status	Meaning
<blank>	A part with the same name and timestamp has not yet been moved to the sandbox. The Duplicate column indicates whether a part with the same name but a different timestamp has been moved to the sandbox.
Processed	A part with the same name and timestamp has been moved to the sandbox but has not yet been committed to ENVY.
Migrated	A part with the same name and timestamp has been moved to the sandbox and committed to ENVY. The sandbox has not been reset from ENVY as described in “Reloading previously migrated ENVY packages back into the MSL Migration Assistance Tool sandbox” on page 72.

Not Found

The part is an associate of a part that was already moved to the sandbox from a different MSL concatenation sequence. This part was not found in that previous concatenation sequence, and thus is identified in the sandbox as a *Not Found* part. You can update the sandbox with this newly found part.

In ENVY Only

The part is in your ENVY workspace, but has not been loaded into the sandbox from either an MSL or from ENVY. This might occur if you have deleted all the packages from the sandbox and have not yet reset the sandbox from your ENVY workspace.

The **Duplicate** column indicates *True* if the part is a duplicate:

- Another part with the same name but a different timestamp is already in the sandbox.
- Another part with the same name is in the same MSL concatenation sequence.

When **Duplicate** is *True*, the **Last Migration Library Timestamp** provides the timestamp of the part in the sandbox to help in determining which of the duplicates you really want to migrate to ENVY.

The value displayed in the **Last Migration Library Timestamp** depends on whether the sandbox has been reloaded from ENVY:

- If the sandbox has not been reloaded (for example, you are working with your initial set of MSLs or continuing with additional MSLs after committing some packages), then the timestamp is the timestamp from the MSL of the part in the sandbox.
- If you have reloaded the sandbox from ENVY (for example, you migrated one subsystem several months ago and now are ready to do the remaining subsystems), then the timestamp is the timestamp of the edition of the part that is currently loaded in your workspace.

The **Last Migration Library Timestamp** is not displayed in the following situations:

- A version of the part is not in the sandbox. This occurs if the duplicates are both in the current MSL concatenation sequence or when you have cleaned out the sandbox, but have not reloaded from your ENVY repository.
- The part in the sandbox is identified as a *Not Found* part. There is no timestamp available for a *Not Found* part.
- The part in the MSL Migration Part List window has the same timestamp as the part in the sandbox.

- The part is not a duplicate.

Other MSL Migration Part List functions

From the MSL Migration Part List window, you can perform the following tasks:

- Add a previously *Not Found* part that you have now found to the sandbox.
- Resolve duplicates by:
 - Removing the duplicate from further consideration so that it no longer appears on the MSL Migration Part List window and will not be considered when looking for associates of other parts.
 - Replacing a part in the sandbox with a different version of the part.
- Add new parts or replace existing parts in packages that are in the sandbox, but which have already been committed to ENVY. This enables you to update an existing ENVY package with a new version of the part.

Working in the sandbox - using the VG Part Prerequisites View window

The VG Part Prerequisites View window shows the ENVY packages that have been created in the sandbox. The number of parts in the package is shown to the right of the package name. When you select one of the packages from the left pane, the parts contained in the package and the required and dependent packages for the package appear in the other panes.

Figure 10 shows the VG Part Prerequisites View window.

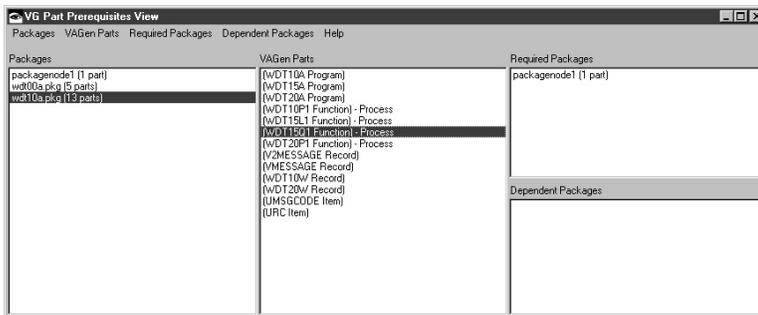


Figure 10. VG Part Prerequisites View window

Identifying common code

The MSL Migration Assistance Tool helps you identify parts that are associated with more than one program. A package.node is created automatically whenever a common part is used by more than one package. The MSL Migration Assistance Tool does this by automatically moving the common part into a new package.node. This removes the common part from the original package. Both the original package and the package currently being created specify the package.node for the common part as a prerequisite

in the **Required Packages** pane. If several parts are shared by two packages, they are all automatically moved to the same package.node. package.nodes are the mechanism that the MSL Migration Assistance Tool uses to prevent putting the same part into two different ENVY packages. package.nodes help you identify common code that might need to be placed in different ENVY packages rather than in the package in which the part was originally placed.

For example, if record ABC is used by Program1, then ABC is initially included in the same package as Program1. Later, if Program2 is placed in a different package in the sandbox and Program2 also uses record ABC, then record ABC will be automatically moved from the package that contains Program1 and placed in a new package.noden, where **n** is a number to distinguish different package nodes.

Identifying missing parts

The MSL Migration Assistance Tool also helps you identify parts that are missing. For example, if you migrate Program1 and it has record DEF defined in its Tables and Additional Records List, record DEF might not exist anywhere in the MSL concatenation sequence. In this case, the record DEF is temporarily included in the same package as Program1, but is identified with the notation *Not Found* in the **VAGen Parts** pane. You can create a package called `my.notfound.pkg` and move the missing parts to that package as you migrate parts to the sandbox. Alternatively, the MSL Migration Assistance Tool moves any missing parts that are included in a package being committed to ENVY to a `notfound.pkg` during the commit process. This is because ENVY does not support empty parts.

You can create a list of the missing parts and write the list to the log file, **mslmig.log**. Then you can use the list to find the missing parts in your MSLs that have not yet been loaded into the sandbox.

Other sandbox functions

From the VG Part Prerequisites View window, you can perform the following tasks:

- Create a new package. If you have not previously created a `common.data.pkg` by moving data items, records, and tables to the sandbox, you might decide to create one to contain shared data items and records that you discover when they are moved to package.nodes as you migrate programs and views.
- Mark a package *unexplodable*. Marking a package *unexplodable* means that if parts in this package are used by a program that is moved to the sandbox later, they are not moved to a new, shared package.node. For example, if you have a `common.data.pkg` that contains all your common data items, records, and tables, you should mark it as *unexplodable* so that these common parts are not continually moved to new package.nodes as you migrate other programs.

- Collapse a package into another package. If you decide that two separate packages would be more useful if they were combined into a single package, the collapse function allows you to merge one package into the other.
- Rename a package if you change your mind about the name.
- Check the consistency of a package to ensure that all required packages are specified.
- Normalize a package to ensure that only the packages that are needed are listed as required packages. This avoids unnecessary entries in the PLP part.
- Change the required packages for a package.
- When you are satisfied with the organization of your packages, you can commit them to ENVY. You are prompted to provide the name of the project in which the selected packages should be stored. If the project does not exist, it is automatically created. If the project does exist, an edition is opened. The commit process then creates the packages, classes, and VAGen parts in ENVY.

If you have modified a package that is already in ENVY, the commit process creates a new edition of the project and package and any affected classes and VAGen parts. New classes and VAGen parts are created within the package as needed.

Reloading previously migrated ENVY packages back into the MSL Migration Assistance Tool sandbox

You might want to migrate one subsystem, work with it for a while in ENVY to gain some experience, and then migrate your other subsystems. If you do this, you need to reset the MSL Migration Assistance Tool from ENVY to reflect the parts that are currently in ENVY. For example, if you added any new parts to ENVY, these new part names might also exist in an MSL that you want to load into the sandbox for migration to ENVY. These parts in the MSL would thus need to be treated as duplicate parts.

Use these steps to load packages into the sandbox from ENVY:

1. From the Workbench window, add into your workspace any packages (such as your common packages) that you need to have in the sandbox when you migrate your new MSLs.
Alternatively, from the Workbench, add the projects that contain the packages you need in your workspace.
2. From the MSL Migration Part List window, select the **ENVY Pkg Selection** push button.

If there are packages already in the sandbox, a message is displayed asking if the packages can be deleted. By deleting the packages currently in the sandbox (from your last migration operation), you can ensure that

you start from a consistent set of packages (all loaded from MSLs or all from ENVY). If you choose not to delete the packages, your new selections are added to the existing sandbox list.

3. After you respond **Yes** or **No** to the message about existing packages, a Selection Required window appears. In this window, select the packages from your workspace that you want to have in the sandbox when you continue your migration with your other subsystems. For example, in most situations, you will want to load your common packages. Loading common packages avoids the MSL Migration Assistance Tool identifying these common parts as *Not Found* when you migrate your other subsystems.
4. From the MSL Migration Part List window, select **MSL Library Selection** to load into the sandbox the next group of MSLs that you want to migrate.

Notes:

1. When packages are loaded into the sandbox from ENVY, the timestamp for their parts in the sandbox reflects the timestamp of the parts in your ENVY workspace, not the timestamp of the original MSL member. Therefore, the **Last Migration Library Timestamp** also reflects the timestamp from ENVY.
2. When packages are loaded into the sandbox from ENVY, there is no analysis to determine if these packages use any parts that have not yet been found.
3. You cannot load a package into the sandbox from your ENVY workspace in the following situations:
 - After migration to ENVY, you added a nonvisual part or a Java class.
 - After migration to ENVY, you added parts for VisualAge Generator control information (generation options, linkage table, resource associations, bind control commands, or linkage editor control statements).

If you try to load packages that have been changed in these ways, you will receive a message saying:

```
Unsuccessful in reading ENVY pkg into tool
```

When you receive this message, the log file (**mstmig.log**) lists the parts that prevent the package from being loaded into the sandbox. To load the package into the sandbox, you must first remove the classes that are causing the problem.

If the package and the class are already versioned, use the following steps:

- a. Open a new edition for the package.
- b. Delete the class.
- c. Load the packages into the sandbox.

- d. Re-add the original package back into your workspace (so that it is in your workspace when you version and release classes after committing the next set of packages in the sandbox).

If the package and the class are not versioned yet, use the following steps:

- a. Version the class.
- b. Delete the class.
- c. Load the package into the sandbox.
- d. Re-add the class (so that it is in your workspace when you version and release classes after committing the next set of packages in the sandbox).

Chapter 11. Migrating production and work-in-progress MSLs to VAGen 4.0 on Java

Your organization might have many MSLs, each used for different purposes. For example, your Database Administrator might be responsible for maintaining one MSL that contains the data item and SQL row record definitions. You might have MSLs that reflect the code that is currently in production, and other MSLs contain the changes that are in the process of being made. These MSLs containing changes include the developers' read/write MSLs, staging MSLs, system test MSLs, and acceptance test MSLs. These MSLs containing changes will be referred to in this document as *work-in-progress* MSLs.

In general, for a system or subsystem, you should migrate your production MSLs first and then migrate your work-in-progress. There are different recommended techniques for migrating production and work-in-progress MSLs.

Migration of your production MSLs involves using the MSL Migration Assistance Tool. The MSL Migration Assistance Tool helps you structure your VisualAge Generator code into ENVY packages by using the sandbox approach explained in "Chapter 10. The MSL Migration Assistance Tool on Java" on page 61. This is easier than using **VAGen Import** for external source format files and then trying to rearrange the parts into ENVY packages.

Migration of your work-in-progress MSLs varies depending on the number of new members in the MSLs:

- If all (or most) members already exist in the production MSLs, you arranged the parts into packages when you migrated your production MSLs. Therefore, to migrate your work-in-progress MSLs you can use **VAGen Import** to import the external source format files directly into ENVY.

VAGen Import allows you to specify that each part in the external source format file is to be placed in the same ENVY package in which it is already defined. This preserves the structure of your ENVY packages that you created using the MSL Migration Assistance Tool.

- If you have many new members that did not exist in the production MSLs, you need to assign these members to packages. For a large number of new members, it is easier to use the MSL Migration Assistance Tool to migrate your work-in-progress MSLs.

The techniques for migrating production and work-in-progress MSLs are explained in:

- “Production MSLs”
- “Work-in-progress MSLs” on page 80

You might want to migrate one subsystem, work with it in ENVY for a while and then migrate your remaining subsystems. To do this, you need to reset the MSL Migration Assistance Tool from ENVY so that any changes you have made to previously migrated parts is reflected in the sandbox. This is described in “Resetting the MSL Migration Assistance Tool sandbox” on page 252.

Production MSLs

Migrating production MSLs involves the following general steps:

1. If you are migrating from Cross System Product, do the following:
 - Export external source format files for each production MSL.
 - Download the external source format files to the workstation.
 - Use the MSL Migration Assistance Tool to create an MSL directory structure on the workstation.
2. If you are migrating from previous versions of VisualAge Generator, your MSLs should already exist on the workstation or LAN. However, to avoid any problems with code page differences, you should:
 - Export external source format files for each production MSL.
 - Convert the external source format files to the Windows NT code page as described in “Changing from OS/2 to Windows NT” on page 119.
 - Use the MSL Migration Assistance Tool to create an MSL directory structure on the workstation.
3. Move parts into the sandbox as described in “Techniques for moving parts to the sandbox” on page 77.
4. Commit the packages in the sandbox to ENVY. When you are prompted for a project name, enter the name of the production-level project (for example, MyProdProject).
5. In ENVY, do the following:
 - Version and release the parts.
 - Version the packages.
 - Create the Project List Parts (PLPs).
 - Version the projects.
 - Generate the programs. Then test to be sure that what you migrated matches your production code.

Techniques for moving parts to the sandbox

There are three basic techniques that can be used to move parts into the sandbox. The choice partly depends on how clean your MSLs are in terms of the following:

- There are no missing parts
- All code is currently used
- There are no duplicates

Each of the three techniques involves using the MSL Migration Assistance Tool and migrating only the MSLs that contain the production level code. The three techniques are described in the following sections:

- “All data items, records, and tables are used”
- “All records and tables are used” on page 78
- “Some records and tables might not be used” on page 78

“Considerations for the migration techniques” on page 79 provides information that applies to all three techniques.

All data items, records, and tables are used

If you know the data items, records, and tables are all actually being used, move the parts to the sandbox in the following order:

1. Data items
2. Records and tables
3. Mark the packages created for data items, records, and tables as *unexplodable*, as described in “Other sandbox functions” on page 71.
4. Programs — their associates that have not yet been moved will be moved to the sandbox with them
5. Any processes, statement groups, functions, maps, map groups, and PSBs that are no longer used or that are not used by the current MSL concatenation.

This technique has the advantage of creating the fewest package.nodes when common parts are discovered (see “Identifying common code” on page 70). Because all data items are migrated first, followed by all records and tables, this technique assumes that all data items, records, and tables are actually used. If this is not the case, there is the disadvantage of migrating parts to ENVY that are never used.

This technique works well if you have an MSL that contains all your common global data items, records, and tables and you know that all these parts are actually used. This might occur if a database administrator maintains the MSL that contains the common global data items, records, and tables.

All records and tables are used

If you know that all records and tables are used, but are not sure whether all the data items are used, move the parts to the sandbox in the following order:

1. Records and tables — their associated data items will be moved to the sandbox with them
2. Mark the package(s) created for the records and tables as *unexplodable*
3. Move any remaining data items to a package called `my.unused.items.pkg`. Do not mark this package as *unexplodable* so items that are used directly by packages (for example, called parameters) will be automatically moved to a new `package.node`. After the data items are identified, you can then move them to the package containing records and tables. What remains in `my.unused.items.pkg` after everything is moved to the sandbox are all the global data items that are no longer used by this MSL concatenation.
If you will be migrating other MSL concatenation sequences that might use these data items, do not commit `my.unused.items.pkg` to ENVY until you have migrated the other MSL concatenations. Waiting to commit enables you to move data items that are used by the later concatenation sequences to other packages in the sandbox.
4. Programs — their associates that have not yet been moved will be moved to the sandbox with them
5. Any processes, statement groups, functions, maps, map groups, and PSBs that are no longer used or that are not used by the current MSL concatenation

This technique has the advantage of minimizing the `package.nodes` that are created automatically. Some data items that are initially thought to be unused might be moved to a `package.node`. For example, this occurs when the only use of a global data item is as a called parameter for a package.

Because records and tables are migrated first, this technique assumes that all records and tables are actually used. If this is not the case, there is the disadvantage of migrating records and tables (and their associated data items) that are never used.

Some records and tables might not be used

If you are not sure whether some records and tables are currently used, move the parts to the sandbox in the following order:

1. Programs — their associates will be moved to the sandbox with them
2. Whatever is left — this might involve data items, records, tables, processes, statement groups, functions, maps, map groups, and PSBs that are no longer used or which are not used by the current MSL concatenation

This technique has the advantage of identifying any parts that are no longer used for this MSL concatenation sequence. It has the disadvantage of potentially creating many package.nodes during migration. This is because data items, records, and tables tend to be shared among many packages. You will be identifying common data items, records, and tables when you move packages to the sandbox rather than taking care of them prior to dealing with programs and GUIs as described in the other techniques.

Considerations for the migration techniques

The following notes apply to all three techniques:

- If you have naming conventions that help you identify common parts, you can use your naming conventions to help you separate parts into ENVY packages. For example, if records starting with V* indicate a common record and records starting with W* belong to a program, you might choose to only move common records (the ones starting with V*) when moving records and tables to the sandbox using the first two techniques.
- The order in which you migrate parts to the sandbox affects how parts are assigned to packages.
- For each MSL or each MSL concatenation sequence, be sure to check that everything in the MSL was moved to the sandbox. Just migrating all the programs does not guarantee that everything in the MSL was migrated — there might be records, tables, data items, and so on that are used by packages in other MSL concatenation sequences but are not used in the current concatenation sequence. You can use the toggle button **Only parts not processed** on the Part List Selection Criteria View window to limit the list of VAGen parts to those that have not yet been processed by the MSL Migration Assistance Tool. For each part that has not been processed, you need to decide whether it needs to be migrated or is no longer used.
- For each MSL concatenation sequence:
 - After committing the packages to ENVY, generate the programs, map groups, and tables. Then test to be sure that what you migrated matches your production code. For programs, you might want do the following:
 - Generate each program without any of its tables
 - Generate each table

This avoids generating common tables multiple times.

If some programs or tables cannot be generated, then this highlights an area where there are problems in migration. To determine whether the problem is in the organization of the ENVY packages or whether the problem is in the original source code, try generating the failing program, map group, or table on the original Cross System Product or VisualAge Generator system using your original MSLs.

Notes:

1. Any programs for the C++ target environments **must** be regenerated for VisualAge Generator 4.0 or later. There is no coexistence of C++ runtime services between VisualAge Generator 2.2 and 4.0 or later. In addition, it is **strongly recommended** that you regenerate all programs for the COBOL environments and package all views to be sure that you migrated the correct level of code.
2. For CICS OS/2, the default *parmform* option in the linkage table was *COMMDATA*. With VisualAge Generator 4.0 or later, the new option *COMMPTR* is the default. Therefore, if you never specified linkage tables for CICS OS/2, you might need a linkage table now.
 - Test the generated programs.

Work-in-progress MSLs

After you have completed the following tasks, you are ready to migrate your work-in-progress MSLs:

- Migrate your production MSLs using the MSL Migration Assistance Tool
- Version and release the parts (views and VAGen part classes)
- Version the packages
- Create Project List Parts (PLPs) for your production level code
- Version the projects
- Generate the programs
- Test your migrated production level code

The process for migrating your work-in-progress MSLs assumes you have separate MSLs for staging, system test, and acceptance test as shown in Figure 11 on page 81.

Depending on the number of new members in your work-in-progress MSLs, use one of the following techniques:

- If there are no (or very few) new members in your work-in-progress MSLs, see “Using VAGen Import” on page 81.
- If there are many new members, see “Using the MSL Migration Assistance Tool” on page 82.

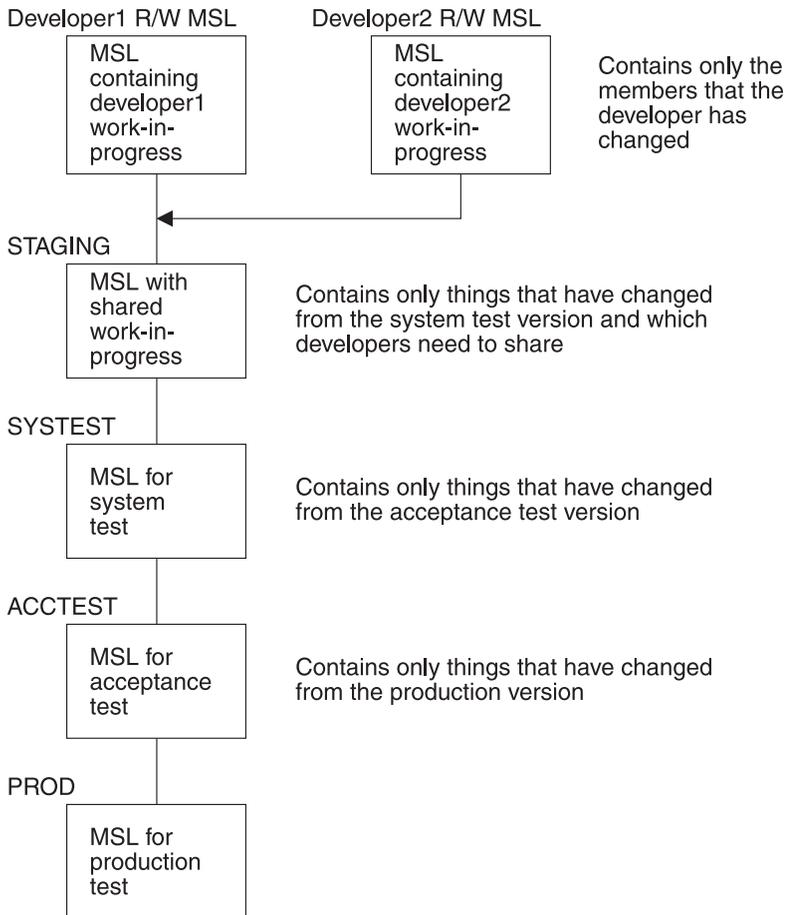


Figure 11. Sample MSL Concatenation for Complete Production MSL and Deltas for Test

Using VAGen Import

If you have no (or very few) new members in your work-in-progress MSLs, you can migrate them using **VAGen Import** by working backward through your MSL concatenation sequence from the production MSLs, and doing the following:

1. Create a test-level project (for example, MyAccTestProj), and add the same packages and versions to the test-level project as are in the production-level project.
2. With the test-level project in your workspace, open editions of the projects and packages so that you will be able to import into them.
3. Export external source format files for the acceptance test (ACCTEST) MSL.

4. Use **VAGen Import** to import the external source format files into ENVY, after selecting the **Defined package** radio button. This causes parts that are in the external source format files that already exist in ENVY to be imported into the same package in which they are already defined.
5. Version and release the classes (VAGen part classes and views).
6. Version the ENVY packages. This provides a base line for the code that was in acceptance test.
7. Create the default package for the current test level (MyAccTestProj). Update the PLP part in the default package, and version the default package.
8. Version the project that contains the version of the packages that are in acceptance test.
9. Generate the programs, map groups, and tables that have changed for the acceptance test level. Then test to be sure that everything migrated successfully at this level.
10. If there are additional levels of test MSLs:
 - a. Create a project for the next level of test (for example, MySysTestProj) and initialize to the same packages and versions as are in the previously migrated test level (in MyAccTestProj).
 - b. Repeat steps 1-8 for each level of testing, working backward through your MSL concatenation sequence. For example, for the MSLs shown in Figure 11 on page 81, you should do the SYSTEST MSL next, followed by the STAGING MSL.
11. After all work-in-progress MSLs have been migrated, other than the developers' read/write MSLs, assign ownership of the projects, packages, and classes.
12. Developers can then migrate their read/write MSLs by doing the following:
 - Open editions of the Development projects and packages.
 - Export external source format files for their read/write MSL.
 - Use **VAGen Import** to import the external source format files into ENVY after selecting the **Defined package** radio button. This causes parts that are in the external source format files that already exist in ENVY to be imported into the same package in which they are already defined.
 - The developers can then continue with their normal development and test work and wait to version and release the classes until their testing is completed.

Using the MSL Migration Assistance Tool

You should create projects for each level of test and seed them from the previous level of test (or production) before migrating the corresponding test-level of MSL using the MSL Migration Assistance Tool.

If you have many new members in your work-in-progress MSLs, it might be easier to use the MSL Migration Assistance Tool so that you have the same flexibility to arrange parts into ENVY packages as you did when you migrated your production MSLs. To migrate your work-in-progress MSLs using the MSL Migration Assistance Tool, you have the following options:

- If you have not made any changes to the parts you have committed to ENVY and your sandbox is still available, you can use the same repository and sandbox that you previously used during migration. The advantage of this is that the **Last Migration Library Timestamp** reflects the timestamp from the MSL(s) that you previously migrated.
- If you have made any changes to the organization of your parts within ENVY or you have added or deleted parts from ENVY, then you must reload the sandbox as described in “Reloading previously migrated ENVY packages back into the MSL Migration Assistance Tool sandbox” on page 72.

All the functions of the MSL Migration Assistance Tool are available to you to assist in migrating your work-in-progress.

Chapter 12. VAGen on Java case studies based on various MSL structures

There are several common ways that you might have organized your MSLs. These include the following techniques:

- Techniques for Production MSLs
 - “Multiple subsystems with no duplicates” on page 87
 - “Multiple subsystems with controlled duplicates” on page 89
 - “Separate production MSLs for each developer” on page 92
 - “MSLs that contain unintended duplicates” on page 97
 - “MSLs containing code from VisualAge Generator Templates or BW*Wizard” on page 98
- Techniques for Work-in-Progress MSLs
 - “Complete set of MSLs for production and deltas for test” on page 102
 - “Complete sets of MSLs for test and production” on page 106
 - “MSLs from marketing or other demonstrations” on page 111

Combinations of the above techniques might be used in your organization. In particular, one of the techniques for production MSLs might be combined with one of the techniques for work-in-progress MSLs.

In addition, because Cross System Product and VisualAge Generator had no formal library management, it is common to find unintended duplicates or even triplicates of members in MSLs. An example of this is described in “MSLs that contain unintended duplicates” on page 97.

Understanding the diagrams and terminology

This chapter uses diagrams to represent MSL concatenation sequences. These diagrams are explained in “MSL structure diagrams”.

This chapter also uses the **advance** command in explaining how some of the MSL concatenation sequences were used. The advance command, which was not supported in Cross System Product, is explained in “Advance command in VisualAge Generator” on page 86.

MSL structure diagrams

Figure 12 on page 86 shows a diagram of an MSL structure similar to the ones that are used in the rest of this chapter.

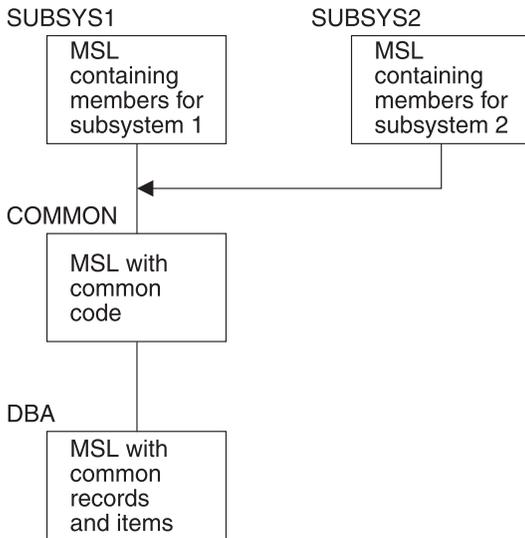


Figure 12. Sample MSL Concatenation

In Figure 12, each box represents a different MSL. The four MSLs are SUBSYS1, SUBSYS2, COMMON, and DBA. The lines between the boxes represent MSL concatenation sequences. Figure 12 shows two MSL concatenation sequences. The first concatenation sequence is SUBSYS1, followed by COMMON and then DBA. The second concatenation sequence is SUBSYS2, followed by COMMON and then DBA.

Advance command in VisualAge Generator

Releases of VisualAge Generator prior to 3.0 provided an **advance** command that was not available in Cross System Product. Advance moves members from one MSL of a concatenation (the source) to the next MSL of the concatenation sequence (the target). After a member is moved to the target MSL, it is deleted from the source MSL. In Figure 12, you can advance a member from SUBSYS1 to COMMON, from SUBSYS2 to COMMON, and from COMMON to DBA. However, you cannot advance a member from SUBSYS1 to DBA.

For Cross System Product, the equivalent function is done by the following steps:

1. Concatenate the target MSL first (read/write) and the source MSL second (read-only).
2. Copy the members from the source MSL to the target MSL.
3. Change the MSL concatenation sequence so the source MSL is first (read/write).
4. Delete the members from the source MSL.

In the sections that follow, advance is used to describe how members are moved from one MSL to the next.

Multiple subsystems with no duplicates

The scenario for separate MSLs for each subsystem has the following MSL structure:

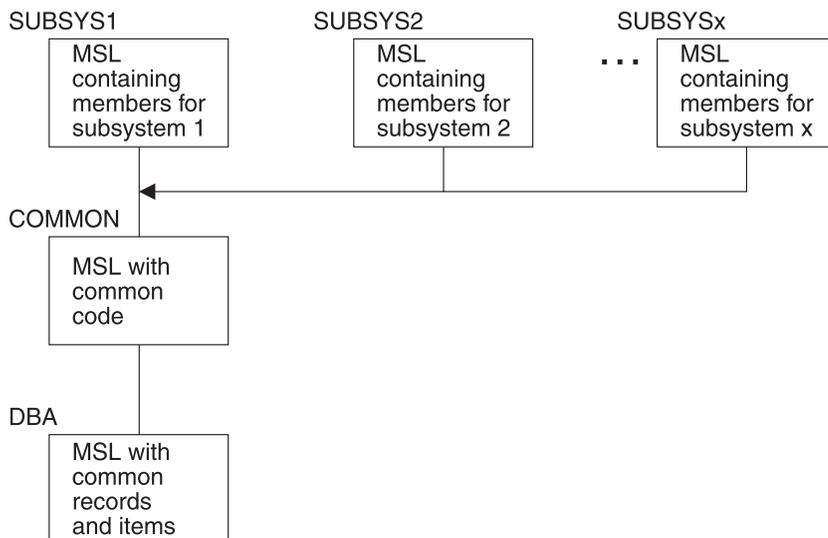


Figure 13. Sample MSL Concatenation for Multiple Subsystems with no Duplicates

In this scenario, there are no duplicate members. Each member exists in one and only one MSL.

Recommendations

Consider migrating the MSLs to ENVY as follows:

1. Verify that there are no duplicates in the MSLs. If there are duplicates, either resolve the duplicates by deleting the obsolete version of the member or see one of the following sections:
 - “Multiple subsystems with controlled duplicates” on page 89
 - “Separate production MSLs for each developer” on page 92
 - “MSLs that contain unintended duplicates” on page 97
2. Move the parts in the DBA and COMMON MSLs to the sandbox. Use a single concatenation sequence for these two MSLs.
3. Commit the packages created from the DBA and COMMON MSLs to ENVY. Specify a project name that reflects that this is common code (for example, CommonProject).
4. Version and release the classes (VAGen part classes).

5. Version the packages created from the DBA and COMMON MSLs.
6. Version the project (CommonProject).
7. Keep the packages created from the DBA and COMMON MSLs in the workspace and the sandbox. This enables the MSL Migration Assistance Tool to specify these packages as required packages where appropriate when migrating the subsystems.
8. Migrate one of the subsystems (for example, SUBSYS1).
9. Commit this group of packages to ENVY. Specify a project name that reflects that this is SUBSYS1 (for example, Subsys1Project).
10. Version and release the classes (VAGen part classes).
11. Create a default package and a Project List Part for Subsystem 1. This Project List Part should specify the project created from the DBA and COMMON MSLs as a required project.
12. Version the packages created from the SUBSYS1 MSL.
13. Version the project (Subsys1Project).
14. Generate the programs moved to ENVY from the SUBSYS1 and COMMON MSLs. Test the code migrated for Subsystem 1.
15. Migrate the next subsystem (SUBSYS2). The packages created from the DBA, COMMON, and SUBSYS1 MSL can all be available in the workspace so they can be specified as prerequisites where appropriate.
16. Commit this group of packages to ENVY. Specify a project name that reflects that this is SUBSYS2 (for example, Subsys2Project).
17. Version and release the classes (VAGen part classes).
18. Create a default package and a Project List Part for Subsystem 2. This Project List Part should specify the project created from the DBA and COMMON MSLs as a required project.
19. Version the packages created from the SUBSYS2 MSL.
20. Version the project (Subsys2Project).
21. Generate the programs moved to ENVY from the SUBSYS2 MSL. You should not need to generate programs in COMMON again unless SUBSYS2 is for a different target environment. Test the code migrated for Subsystem 2.
22. Migrate the work-in-progress MSLs using one of the following techniques:
 - “Complete set of MSLs for production and deltas for test” on page 102
 - “Complete sets of MSLs for test and production” on page 106
 - “MSLs from marketing or other demonstrations” on page 111

Because there are no duplicate parts, the projects for both SUBSYS1 and SUBSYS2 can be loaded into the same workspace.

Multiple subsystems with controlled duplicates

The scenario for separate MSLs for each subsystem, but with controlled duplicates has the following MSL structure:

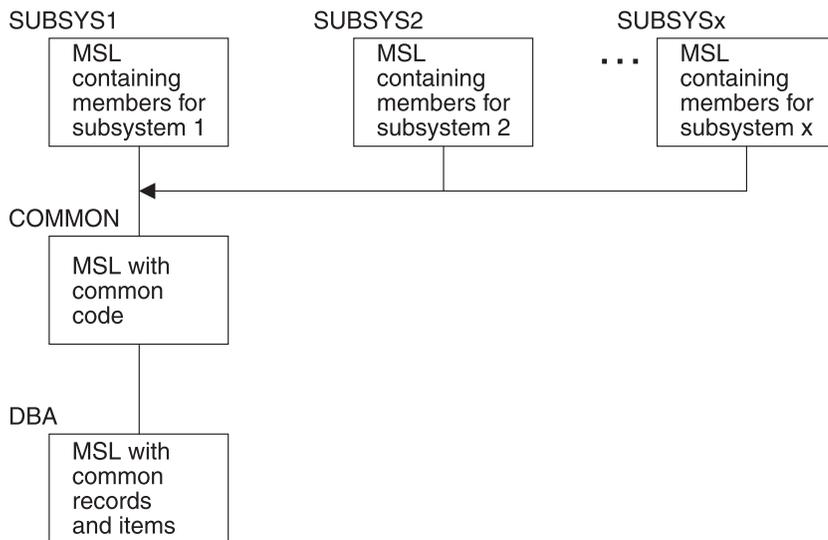


Figure 14. Sample MSL Concatenation for Multiple Subsystems with Controlled Duplicates

Although the scenario in Figure 14 appears the same as the scenario in Figure 13 on page 87, the Figure 14 scenario is different in that the same member might exist in the SUBSYS1, SUBSYS2, and SUBSYSx MSLs. For example, the COMMON MSL might contain a message handling program that obtains the messages from a VisualAge Generator table. Each subsystem has its own VisualAge Generator message table.

This results in duplicates, but only intentional duplicates, between the subsystems. There are no duplicates between a subsystem and the COMMON MSL.

For CICS and IMS target environments, this assumes that the subsystems run in separate regions — duplicate program, table, or map group names are not permitted in the same region.

Recommendations

Consider migrating the MSLs to ENVY as follows:

1. Determine the list of members that have been intentionally duplicated between the subsystems. Ensure that there are no duplicates between the DBA or COMMON MSLs and any of the subsystems.

2. Move the parts in the DBA and COMMON MSLs to the sandbox. Use a single concatenation sequence for these two MSLs. There might be a number of parts marked as *Not Found* because each subsystem contains its own version of the part. Be sure that these parts are included in the subsystem MSLs. For example, if each subsystem has its own message table, the table would not be in either the DBA or COMMON MSL and would be reported as *Not Found* when you migrate these two MSLs.
3. Commit the packages created from the DBA and COMMON MSLs to ENVY. Specify a project name that reflects that this is common code (for example, CommonProject).
4. Version and release the classes (VAGen part classes).
5. Version the packages created from the DBA and COMMON MSLs.
6. Version the project (CommonProject).
7. Keep the packages created from the DBA and COMMON MSLs in the workspace and the sandbox. This enables the MSL Migration Assistance Tool to specify these packages as required packages where appropriate when migrating the subsystems.
8. Migrate one of the subsystems (for example, SUBSYS1).
 - Some of the intentional duplicates might have a **Status** of *Not Found* on the MSL Migration Part List window. For example, if an error handling program from COMMON uses different message tables for each subsystem, the message table would be identified as a *Not Found* part until you migrate it to the sandbox for the first subsystem.
 - Put the intentional duplicates for this subsystem into one or more packages that have a name that identifies them as duplicates. For example:


```
xxx.yyyyyy.zzzzzz
```

where:

```
xxx
```

 Is the subsystem ID.

```
yyyyyy
```

 Is something that indicates a duplicate (such as duplicate, overwrite, special, subsystem).

```
zzzzzz
```

 Is what is being duplicated (such as messages, menu.table, helptext).
 - Mark the packages created for the intentional duplicates as unexplodable.
 - Move the other parts from this MSL concatenation to the sandbox.
9. Commit this group of packages to ENVY. Specify a project name that reflects that this is SUBSYS1 (for example, Subsys1Project). Include the packages created for this subsystem's intentional duplicates in the project.

10. Version and release the classes (VAGen part classes).
11. Create a default package and a Project List Part for Subsystem 1. This Project List Part should specify the project created from the DBA and COMMON MSLs as a required project.
12. Version the packages created from the SUBSYS1 MSL.
13. Version the project (Subsys1Project).
14. Generate the programs moved to ENVY from the SUBSYS1 and COMMON MSLs. Test the code migrated for Subsystem 1.
15. Prepare to migrate the next subsystem by doing the following:
 - From the Workbench window, delete the project created for SUBSYS1 from the workspace. Keep the project created from DBA and COMMON in the workspace.
 - From the VG Part Prerequisites View window, delete the packages created for SUBSYS1 from the VG Part Prerequisites View window so that duplicates between Subsystem 1 and Subsystem 2 will not be detected. Before you can delete a package, you must first remove it from the **Required Packages** list of all other packages in the sandbox.
16. Move the next subsystem (SUBSYS2) to the sandbox. Use the same process that is described in 8 on page 90.
17. Commit this group of packages to ENVY. Specify a project name that reflects that this is SUBSYS2 (for example, Subsys2Project). Include the packages created for this subsystem's intentional duplicates in the project.
18. Version and release the classes (VAGen part classes).
19. Create a default package and a Project List Part for Subsystem 2. This Project List Part should specify the project created from the DBA and COMMON MSLs as a required project.
20. Version the packages created from the SUBSYS2 MSL.
21. Version the project (Subsys2Project).
22. Generate the programs moved to ENVY from the SUBSYS2 MSL. If the migration of Subsystem 2 replaced any parts used by the COMMON code, you would need to generate the programs created from COMMON again. For example, if each subsystem has its own version of Record1, then any program created from COMMON that used Record1 would need to be generated for Subsystem 2. Test the code migrated for Subsystem 2.
23. Migrate the work-in-progress MSLs using one of the following techniques:
 - "Complete set of MSLs for production and deltas for test" on page 102
 - "Complete sets of MSLs for test and production" on page 106
 - "MSLs from marketing or other demonstrations" on page 111

Either the project for SUBSYS1 or for SUBSYS2 can be added into a workspace, but both cannot be added at the same time.

Separate production MSLs for each developer

The scenario for separate production MSLs for each developer has the following MSL structure:

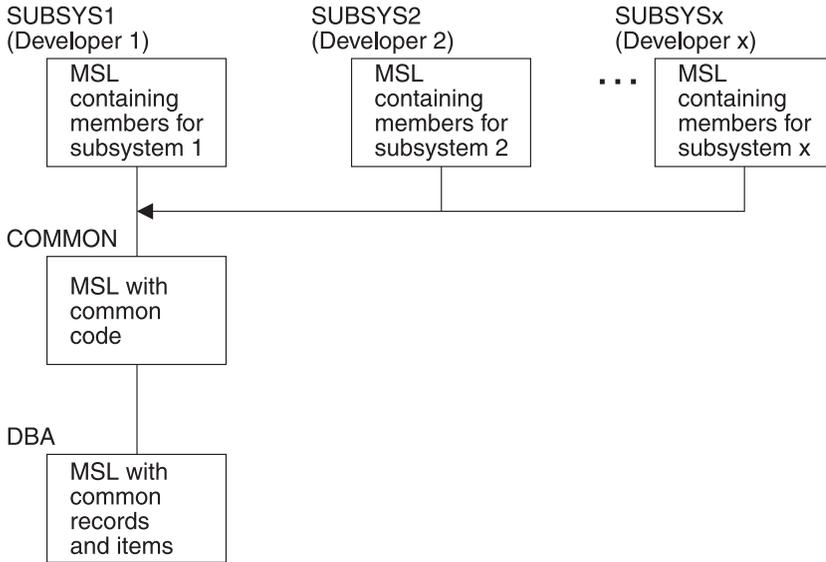


Figure 15. Sample MSL Concatenation when Each Subsystem Replaces Common Code

Although the scenario in Figure 15 appears very similar to Figure 14 on page 89, the situation in Figure 15 is different in that each subsystem was written by a different developer without regard to what the other developers were doing. In this scenario, the same member can exist in the COMMON, SUBSYS1, and SUBSYS2 MSLs. This is because developers are free to modify the common code to suit their own needs. Thus some developers might use the common member unchanged (and still in COMMON) and other developers might each have a different version of the common member. This results in lots of duplicate members, all at different and very likely conflicting levels of code.

There should be a core set of common code, without duplicates in any MSL. However, the following situation could also arise:

SUBSYS1 MSL - members A, B, C	from COMMON: D
SUBSYS2 MSL - members B, C	from COMMON: A, D
SUBSYS3 MSL - members A	from COMMON: B, C, D
SUBSYS4 MSL - members C	from COMMON: A, B, D
COMMON MSL - members A, B, C, D	

In this situation, the COMMON MSL would need to create the following packages:

```
CommonCore: part D
CommonA:    part A
CommonB:    part B
CommonC:    part C
```

As you can see from this small example, determining how the members could be partitioned efficiently might be quite difficult.

Recommendations

How you migrate when each subsystem has been able to modify the COMMON and DBA code depends on whether you want to define a core set of code that no one is allowed to modify or whether you want each subsystem to have its own version of all code and thus work on a stand-alone basis.

Creating a core common set of packages

If you want to change the philosophy that each developer can modify the common code, consider migrating the MSLs into ENVY as follows:

1. Determine the core set of members from the COMMON and DBA MSLs that none of the developers have modified for any subsystem.
2. Build an external source format file containing only this core set of members.
3. Use the MSL Migration Assistance Tool to create an MSL called CORE for this external source format file.
4. Build a second external source format file containing all the members from COMMON and DBA that are not in the core set (members that are changed in one or more subsystems).
5. Use the MSL Migration Assistance Tool to create an MSL called CHANGED for this external source format file.
6. Move the parts from the CORE MSL to the sandbox. There might be a number of parts marked as *Not Found* because individual subsystems have modified these parts.
7. Commit the packages created from the CORE MSL to ENVY. Specify a project name that reflects that this is the core common code (for example, CoreCommonProject).
8. Version and release all the classes (VAGen part classes).
9. Version the packages.
10. Version the project (CoreCommonProject).
11. Keep the packages created from the CORE MSL in the workspace and the sandbox. This enables the MSL Migration Assistance Tool to specify these packages as required packages where appropriate when migrating the

subsystems. This also avoids associated parts being marked as *Not Found* if they are part of the core common code.

12. Move the parts for the SUBSYS1 MSL into the sandbox. Include the CHANGED MSL at the bottom of the concatenation sequence. This enables you to locate any associated parts that are not included in the CORE MSL, but which were not modified for SUBSYS1. Do not add or replace any parts in the packages created from the CORE MSL.
13. Commit the packages created from the SUBSYS1 and CHANGED MSLs to ENVY. Specify a project name that reflects that this is SUBSYS1 (for example, Subsys1Project). This project represents the parts that existed for Subsystem 1.
14. Version and release all the classes (VAGen part classes).
15. Create a default package and a Project List Part for Subsystem 1. This Project List Part should specify the project created from the CORE MSL as a required project.
16. Version the packages.
17. Version the project (Subsys1Project).
18. Generate the programs moved to ENVY for Subsystem 1. Test the code migrated for Subsystem 1.
19. Prepare to migrate the next subsystem by doing the following:
 - From the **Workbench** window, delete the project created for SUBSYS1 and CHANGED from the workspace. Keep the project created from the CORE MSL in the workspace.
 - From the VG Part Prerequisites View window, delete the packages created for SUBSYS1 and CHANGED from the VG Part Prerequisites View window so that duplicates between Subsystem 1 and Subsystem 2 will not be detected. Before you can delete a package, you must first remove it from the **Required Packages** list of all other packages in the sandbox.
20. Move the next subsystem (SUBSYS2) to the sandbox. Include the CHANGED MSL at the bottom of the concatenation sequence. This enables you to locate any associated parts that are not included in the CORE MSL, but which were not modified for SUBSYS2.

Be sure the project created from the CORE MSL is loaded into the workspace and the sandbox. This avoids associated parts being marked as *Not Found* if they are part of the core common code. Do not add or replace any parts in the packages created from the CORE MSL.
21. Commit the packages created from the SUBSYS2 and CHANGED MSLs to ENVY. Specify a project name that reflects that this is SUBSYS2 (for example, Subsys2Project). This project represents the parts that existed for Subsystem 2.
22. Version and release all the classes (VAGen part classes).

23. Create a default package and a Project List Part for Subsystem 2. This Project List Part should specify the project created from the CORE MSL as a required project.
24. Version the packages.
25. Version the project (Subsys2Project).
26. Generate the programs moved to ENVY for Subsystem 2. If the migration of Subsystem 2 replaced any parts used by the CORE code, you would need to generate the programs created from CORE again. For example, if each subsystem has its own version of Record1, then any program created from CORE that used Record1 would need to be generated for Subsystem 2. Test the code migrated for Subsystem 2.
27. Migrate the work-in-progress MSLs using one of the following techniques:
 - “Complete set of MSLs for production and deltas for test” on page 102
 - “Complete sets of MSLs for test and production” on page 106
 - “MSLs from marketing or other demonstrations” on page 111

Creating stand-alone subsystems

If you want each subsystem to have its own version of all code so its developers can work on a stand-alone basis, consider migrating your MSLs to ENVY as follows:

1. For the first subsystem, move parts to the sandbox using a concatenation sequence of SUBSYS1, COMMON, and DBA. The concatenation sequence should be the same concatenation sequence that you use to generate programs for Subsystem 1.
 If there are parts in COMMON and DBA that are not currently used by Subsystem 1, consider putting them into a package called xxx.unused.parts.pkg, where xxx is the subsystem ID. Using this technique means that the common definition of these unused parts will be available in the project for Subsystem 1.
2. Commit the packages created for Subsystem 1 to ENVY. Specify a project name that reflects that this is SUBSYS1 (for example, Subsys1Project). This project represents the parts that existed for Subsystem 1. Include xxx.unused.parts.pkg with the packages you commit to ENVY. The other subsystems will develop their own lists of unused parts.
3. Version and release all the classes (VAGen part classes).
4. Version the packages.

Note: A Project List Part is not needed because all the parts used by Subsystem 1 are in a single project.
5. Version the project (Subsys1Project).
6. Generate the programs moved to ENVY for Subsystem 1. Test the code migrated for Subsystem 1.

7. Prepare to migrate the next subsystem by doing the following:
 - From the Workbench window, delete the project created for Subsystem 1 from the workspace.
 - From the VG Part Prerequisites View window, delete all the packages created for Subsystem 1, including the packages created from the COMMON or DBA MSLs and the xxx.unused.parts.pkg
8. Move the next subsystem to the sandbox using a concatenation sequence of SUBSYS2, COMMON, and DBA. The concatenation sequence should be the same concatenation sequence that you use to generate programs for Subsystem 2.

Note: This technique means that identical code for a part might exist in the packages created for Subsystem 1 and Subsystem 2. This occurs if both of these subsystems used the version of the part that was previously in COMMON or DBA. However, some other subsystem might have modified the part or the developers for Subsystem 1 might need to have their own version of the part in the future. This technique enables the developers of Subsystem 1 and Subsystem 2 to continue to develop in an independent manner, without regard to how the other subsystem uses the part.

9. Commit the packages created for Subsystem 2 to ENVY. Specify a project name that reflects that this is SUBSYS2 (for example, Subsys2Project). This project represents the parts that existed for Subsystem 2. Include xxx.unused.parts.pkg with the packages you commit to ENVY.
10. Version and release all the classes (VAGen part classes).
11. Version the packages.

Note: A Project List Part is not needed because all the parts used by Subsystem 2 are in a single project.

12. Version the project (Subsys2Project).
13. Generate the programs moved to ENVY for Subsystem 2. Because each subsystem has its own complete set of code, you should generate all the programs for Subsystem 2. Test the code migrated for Subsystem 2.
14. Migrate the work-in-progress MSLs using one of the following techniques:
 - “Complete set of MSLs for production and deltas for test” on page 102
 - “Complete sets of MSLs for test and production” on page 106
 - “MSLs from marketing or other demonstrations” on page 111

MSLs that contain unintended duplicates

This scenario contains unintended duplicates. There are multiple subsystems, with the following **intended** MSL structure:

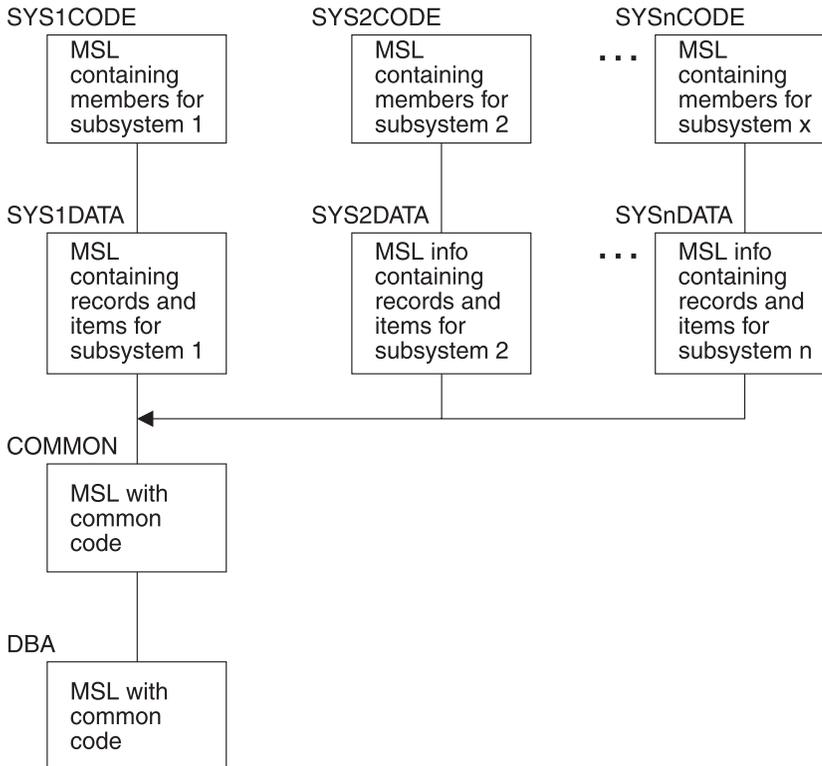


Figure 16. Sample MSL Concatenation when Using VisualAge Generator Templates

Each member was intended to be in one and only one MSL. However, the MSL structure contained the following:

- Some members were never advanced to the production MSLs; they were still in the developer's read/write MSLs even though the changes were complete and were in the production load libraries.
- Some members that were shared between subsystems were not contained in the COMMON MSL; they were in only one subsystem's MSLs. For example, this occurred for records and their associated global data items that were passed when transferring between two subsystems. In this case the real concatenation sequence was SYS1CODE, SYS1DATA, SYS2DATA, and COMMON. In some cases several additional data MSLs were required to generate a subsystem.

- Applications, processes, statement groups, and maps existed both in the SYSnCODE and SYSnDATA MSL — with the same names. This was because they were accidentally put into the wrong MSLs.
- The same members existed in multiple subsystems. For example, the same process might appear in SYS1CODE, SYS3CODE and SYSnCODE.
- Two subsystems used a different COMMON MSL, called COMMON2. COMMON2 contained 10 - 20 programs, as well as other members, that were also included in COMMON.

Recommendations

Before attempting to migrate this set of MSLs, do the following:

- Be sure you know the real MSL concatenation sequences that are required for generation. This will help to minimize the number of *Not Found* members during migration. One possibility is to validate existing applications in the MSLs prior to attempting migration to determine which, if any, members are missing.
- Try to resolve as many duplicates as possible. One possibility is to use the MSL Migration Assistance Tool for a trial migration to determine what duplicates exist and which members are missing. Then go back to the MSLs, make any corrections required to resolve duplicates, restore the missing members, and correct any other errors in the parts.
- After you have resolved the problems, then migrate using the scenario that best matches your new environment.

MSLs containing code from VisualAge Generator Templates or BW*Wizard

The scenario for VisualAge Generator Templates or BW*Wizard has the following MSL structure:

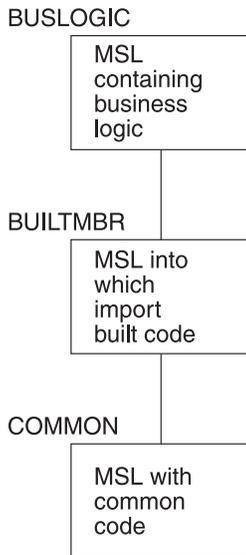


Figure 17. Sample MSL Concatenation when Using VisualAge Generator Templates or BW*Wizard

Programs built with VisualAge Generator Templates or BW*Wizard are imported into the MSL called BUILTMBR. Then the BUSLOGIC read/write MSL is concatenated in front of the read-only BUILTMBR MSL and all *business logic* changes are made in BUSLOGIC. The advantages of this technique are:

- The program can be built again from the template and imported into the BUILTMBR MSL without destroying the business logic.
- If necessary, a comparison of the newly built member in BUILTMBR with the old business logic version of the member can be done to determine if any business logic changes are required to incorporate function that has been added to the newly built member.

This same pairing of a business logic MSL and “built member” MSL can be extended from the developer’s MSLs to the staging, test, and production MSLs. In some cases, the separation of the business logic might be used for the developer’s, staging, and test MSLs, but not be used for the production MSLs.

The COMMON MSL contains any members that are used across subsystems. There might also be a DBA MSL that contains data item definitions.

This example assumes that all the code in BUILTMBR and BUSLOGIC are for a single subsystem called Subsystem 1. If you have multiple subsystems, you would need to extrapolate the example for your situation.

Recommendations

Consider migrating the MSLs to ENVY as follows:

1. Move the parts from the COMMON MSL to the sandbox.

Note: If there is a DBA MSL, migrate it with the COMMON MSL in a single concatenation sequence.

2. Commit the packages created from the COMMON MSL (and DBA MSL if used) to ENVY. Specify a project name that reflects that this is common code (for example, CommonProject).
3. Version and release all the classes (VAGen part classes). You can use **1.0** (the default) as the version number.
4. Version the packages. You can use **1.0** (the default) as the version number.
5. Version the project (CommonProject).
6. Keep the packages created from the COMMON MSL (and DBA MSL if used) in the workspace and the sandbox. This enables the MSL Migration Assistance Tool to specify these packages as required packages where appropriate when migrating the subsystems.
7. Move parts from the BUILTMBR MSL to the sandbox.
8. Commit the packages created from the BUILTMBR MSL to ENVY. Specify a project name that reflects that this is Subsystem 1 (for example, Subsys1Project).
9. Version and release all the classes (VAGen part classes). Name the version to indicate that it is for parts built by the templates. For example, you could use `Built-1.0`.
10. Create a default package and a Project List Part for Subsystem 1. This Project List Part should specify the project created from the COMMON MSL (and DBA MSL if used) as a required project.
11. Version the packages. Name the version to indicate it is for parts built by the templates. For example, you could use `Built-1.0`. Versioning the package enables you to reload the level of code that was built by the template without having to load each class individually.
12. If you might frequently need to load the level of code built from the templates, version the project, using a version of `Built-1.0`.
13. Keep all the packages from the COMMON and BUILTMBR MSLs in the sandbox and in your workspace. This enables you to replace parts in the packages created from the BUILTMBR MSL.
14. Move the parts in the BUSLOGIC MSL to the sandbox. You might have two types of parts in the business logic MSL:
 - Parts that you added for business logic that were not originally built by the templates. These parts will have blanks in the **Duplicate** and **Last Migration Library Timestamp** columns. You can create new

packages for these parts or add them into an existing (committed) package. If you add or change parts in an existing package, the MSL Migration Assistance Tool marks the package as *Modified* and you will be able to commit the package to ENVY again to put the new parts into the ENVY library manager.

- Parts that were originally built for the templates that you modified for business logic. These parts will have *True* in the **Duplicate** column and the timestamp from the template-built MSL in the **Last Migration Library Timestamp** column. Move these parts to the sandbox using **Handle Duplicate Parts** and specifying that they should **Replace Existing** parts. This causes each part to be placed in the same package where it already exists. The MSL Migration Assistance Tool marks the package as *Modified* and you will be able to commit the package to ENVY again to put the new parts into the ENVY library manager.

This technique preserves the package organization you created when you migrated the BUILTMBR MSL, but to include the business logic version of existing parts and any new parts that were not created by the templates.

Note: If you do not have any new parts created for business logic in the BUSLOGIC MSL, you can:

- Create an external source format file for the BUSLOGIC MSL.
 - Use **VAGen Import** to import the external source format file and create new editions of the changed parts. Select the **Defined package** radio button to put the changed editions of the parts into the packages in which they are already defined.
15. Commit any packages that were created or modified for the BUSLOGIC MSL to ENVY. Specify the same project name that you used when committing the BUILTMBR parts (Subsys1Project). This project now represents the actual system that was contained in the concatenation sequence for the BUSLOGIC, BUILTMBR, and COMMON MSLs.
 16. Version and release all the classes (VAGen part classes). Name the version to indicate that it is for parts changed for business logic. For example, you could use `BusLogic-1.1`.
 17. You should not need to change the Project List Part because the `CommonProject` has not changed.
 18. Version the packages. Name the version to indicate it is for parts changed for business logic. For example, you could use `BusLogic-1.1`.

Note: Not all of the classes or packages will have new versions created when you migrate the BUSLOGIC MSL. Only those classes and packages modified for business logic will have a new version.

19. Version the project (Subsys1Project) using version `BusLogic-1.1`.

20. Generate the programs moved to ENVY. Test the code.
21. Migrate the work-in-progress MSLs using one of the following techniques:
 - “Complete set of MSLs for production and deltas for test”
 - “Complete sets of MSLs for test and production” on page 106
 - “MSLs from marketing or other demonstrations” on page 111

You will need to adapt these techniques to capture both the parts built from the templates and the parts that have business logic.

Special considerations for VisualAge Generator Templates

Make sure you change all Statement Group part names to uppercase before migration. The VisualAge Generator Templates product generates Statement Group part names as either all lowercase or mixed case. However, VisualAge Generator searches for these part names in all uppercase (when generating runtime code) and cannot find the parts when the names are in mixed case or lowercase.

Complete set of MSLs for production and deltas for test

This scenario has the following MSL structure. The Database Administrator (DBA) MSLs might be combined with the corresponding STAGING, SYSTEST, or PROD MSL depending on your environment.

Note: There might be one series of the MSLs shown in Figure 18 on page 103 for each subsystem.

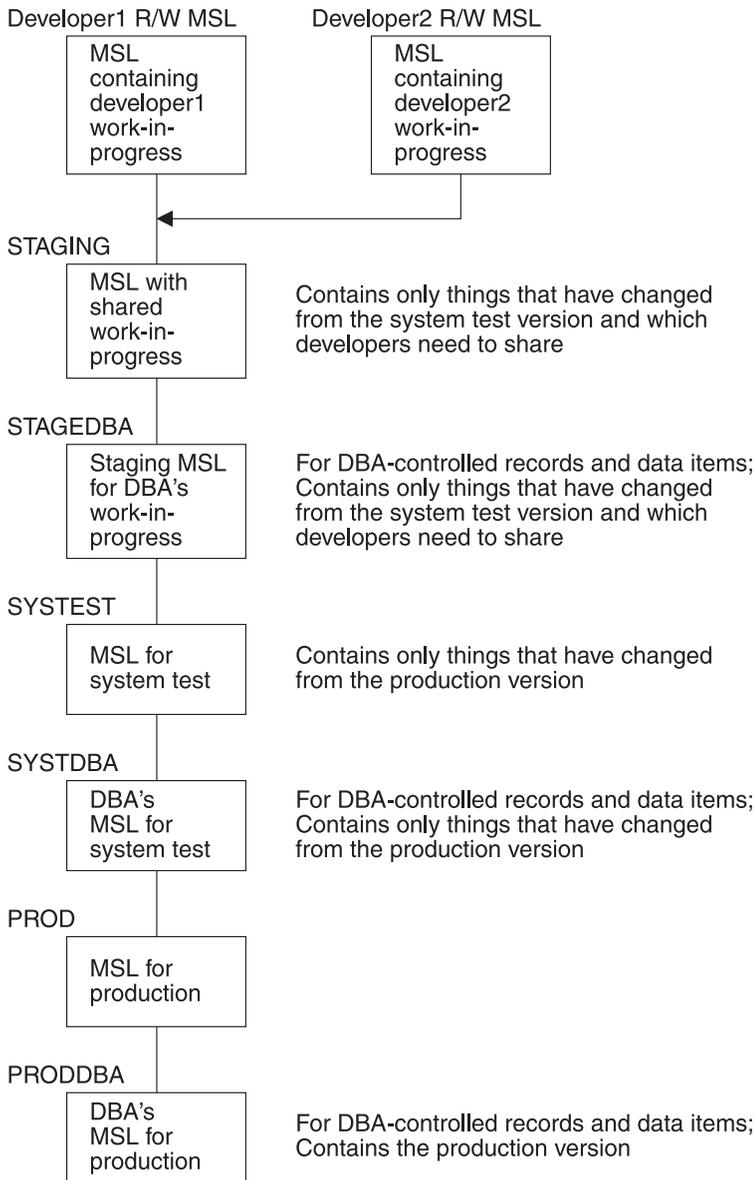


Figure 18. Sample MSL Concatenation for Complete Production MSLs and Deltas for Test

The PROD and PRODDBA MSLs are the only complete pair of MSLs. In other words, this is the only pair of MSLs in which all members exist. SYSTEST and SYSTDBA contain only the members that have been added or changed and which are undergoing system test. STAGING and STAGEDBA contain only the members that have been added or changed and which are still in development testing, but which need to be shared by multiple developers.

Two additional MSL concatenation sequences are used for advancing (moving) members through the MSL hierarchy.

DEVnMSL+STAGING+SYSTEST+PROD - to advance members not controlled by DBA

DEVDBA+STAGEDBA+SYSTDBA+PRODDBA - to advance members controlled by DBA

As newly added or changed members progress through testing, they advance from the developer's read/write MSL to the staging MSL, then from staging to the appropriate development level MSL, then from development to the corresponding system test MSL, and finally from system test to the corresponding production level MSL. Thus the developer read/write, staging, and system test MSLs represent work-in-progress. Similarly, the DEVDBA, STAGEDBA, and SYSTDBA MSLs represent work-in-progress for the DBA.

If there are multiple subsystems, each has its own staging, system test, and production MSLs. There are no duplicate members between the subsystems.

Recommendations

Consider migrating the MSLs to ENVY as follows:

1. Migrate the production MSLs by doing the following:
 - Load the production level MSLs using the MSL Migration Assistance Tool. When you commit to ENVY, specify a project name that indicates that this is production level code (for example, xxxProductionProject, where xxx is the subsystem ID).
 - Version and release the changed classes (VAGen part classes).
 - Version the packages.
 - Create a default package and a Project List Part for the production project (xxxProductionProject).
 - Version the project (xxxProductionProject).
 - Generate the programs. Then test to be sure that what you migrated matches your production code.
2. Migrate the system test MSLs by doing the following:
 - Delete the xxxProductionProject from your workspace.
 - Create a system test level project with a name that reflects that this is the system test level of code (for example, xxxSysTestProject, where xxx is the subsystem ID). Include the same version of all the packages from the xxxProductionProject. Add the xxxSysTestProject to your workspace.
 - Version the system test project (xxxSysTestProject) and then open a new edition of it.
 - Open new editions of the packages.
 - Export an external source format file for the SYSTEST MSL and another external source format file for the SYSTDBA MSL.

- Use **VAGen Import** to import the external source format files and create new editions of the changed parts. Select the **Defined package** radio button to put the changed editions of the parts into the package(s) in which they are already defined.
 - Version and release the changed classes (VAGen part classes).
 - Version any packages that had new editions of classes.
 - Create a default package and a Project List Part for the system test project (xxxSysTestProject).
 - Version the project (xxxSysTestProject).
 - Generate the programs for the system test level of code. You should only need to generate programs, map groups, and tables that have changed. Then test to be sure that what you migrated matches your system test level of code.
3. Migrate the staging MSLs by doing the following:
 - Delete the xxxSysTestProject from your workspace.
 - Create a staging level project with a name that reflects that this is the staging level of code (for example, xxxStagingProject, where xxx is the subsystem ID). Include the same version of all the packages from the xxxSysTestProject. Add the xxxStagingProject to your workspace.
 - Version the staging project (xxxStagingProject) and then open a new edition of it.
 - Open new editions of the packages.
 - Export an external source format file for the STAGING MSL and another external source format file for the STAGEDBA MSL.
 - Use **VAGen Import** to import the external source format files and create new editions of the changed parts. Select the **Defined package** radio button to put the changed editions of the parts into the packages in which they are already defined.
 - Version and release the changed classes (VAGen part classes).
 - Version any packages that had new editions of classes.
 - Create a default package and a Project List Part for the staging project (xxxStagingProject).
 - Version the project (xxxStagingProject).
 - Generate the programs for the staging level of code. You should only need to generate programs, map groups, and tables that have changed. Then test to be sure that what you migrated matches your staging level of code.
 4. Assign ownership of the classes, packages, and projects.
 5. Developers can then load their own work-in-progress doing the following:
 - Export an external source format file for their own read/write MSL.

- Load the project for the staging level of code (xxxStagingProject) into their workspace.
- Use **VAGen Import** to import the external source format files and create new editions of the changed parts. Select the **Defined package** radio button to put the changed editions of the parts into the packages in which they are already defined.

Note that this scenario is easier to load into ENVY than “Complete sets of MSLs for test and production” because the added/changed members for the system test MSL are already determined (everything in the SYSTEST MSL) and because there is no development level set of MSLs to load.

Complete sets of MSLs for test and production

This scenario has the following MSL structure:

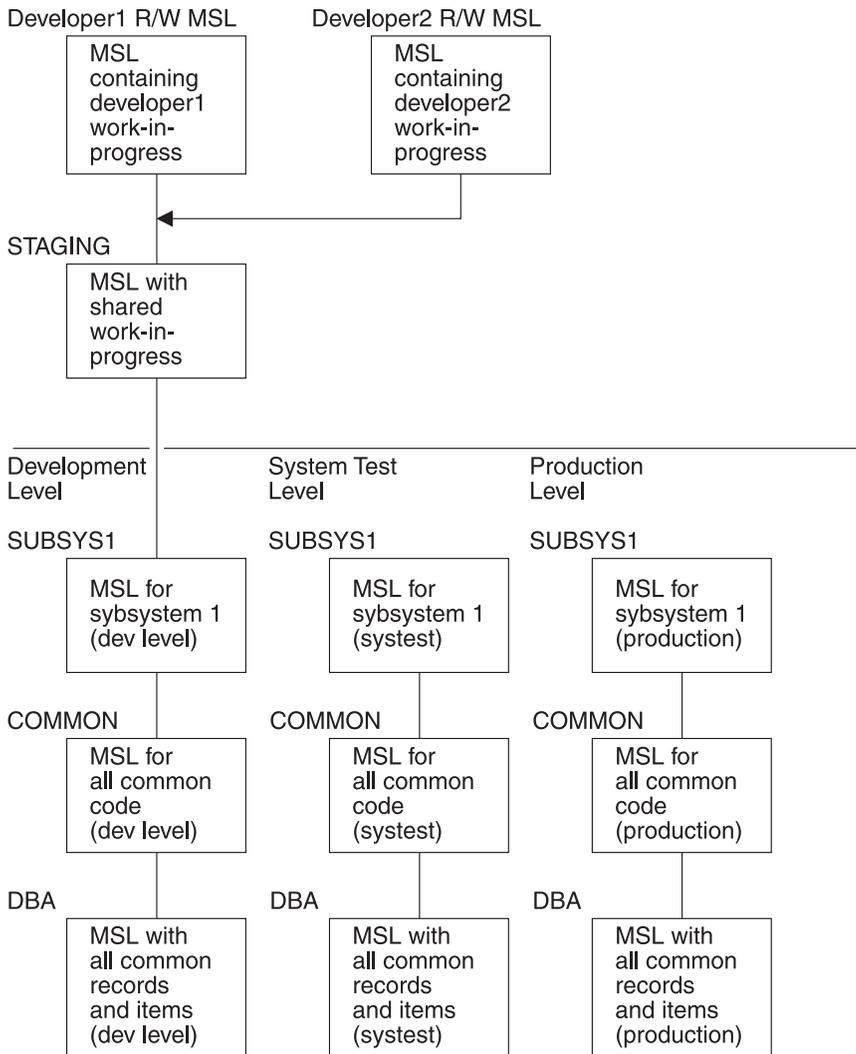


Figure 19. Sample MSL Concatenation for Complete Sets of Development, System Test, and Production

This scenario differs from that described in “Complete set of MSLs for production and deltas for test” on page 102 in that in Figure 19 the SUBSYS1, COMMON, and DBA MSLs at each of the development, system test, and production levels are complete sets of MSLs. In other words, most members exist in all three MSL concatenation sequences with the same date/time stamp. If a member is in the COMMON development MSL, it is also in the COMMON system test MSL, and the COMMON production MSL.

As members progress through testing, the added or changed members advance from the developer’s read/write MSL to the staging MSL and then

from staging to the appropriate development level MSL. As further testing occurs, members are **copied (not moved or advanced)** from the development level MSL to the corresponding system test MSL and then from the system test MSL to the corresponding production MSL.

The developer read/write, staging, development level MSLs, and system test MSLs represent work-in-progress. However, the development level MSLs and the system test MSLs are not true work-in-progress. For example, the system test SUBSYS1 MSL contains the same members as the production SUBSYS1 MSL. Most of the members are identical between system test and production. Perhaps only 5% of the members in the system test SUBSYS1 MSL are currently undergoing system test and represent different versions of the members from what is in the production SUBSYS1 MSL. Similarly, the development level SUBSYS1 MSL for the most part duplicates the members in the system test and production SUBSYS1 MSLs. Perhaps another 5% of the members differ from what is in the system test MSL and represent work that is in development testing (not yet put into system test).

In the same way, the COMMON and DBA MSLs are virtually identical at the development, system test, and production levels. Because the members in these MSLs change less frequently, in many cases there might be **no** members that differ between the three levels of these MSLs.

This means that there are many members in the development level and system test MSLs that should not really be put into ENVY because they are no different than the members that will be migrated using the production MSLs.

Note: If there are multiple subsystems, they can share the COMMON and DBA MSLs, but each subsystem has its own SUBSYSn MSL, where n is the subsystem number. The SUBSYSn MSLs do not have any duplicate member names.

Recommendations

Consider migrating the MSLs to ENVY as follows:

1. Migrate the production MSLs by doing the following:
 - Load the production level MSLs using the MSL Migration Assistance Tool. When you commit to ENVY, specify a project name that indicates that this is production level code (for example, xxxProductionProject, where xxx is the subsystem ID).
 - Version and release the changed classes (VAGen part classes).
 - Version the packages.
 - Create a default package and a Project List Part for the production project (xxxProductionProject).
 - Version the project (xxxProductionProject).

- Generate the programs. Then test to be sure that what you migrated matches your production code.
2. Migrate the system test MSLs by doing the following:
 - Delete the xxxProductionProject from your workspace.
 - Create a system test level project with a name that reflects that this is the system test level of code (for example, xxxSysTestProject, where xxx is the subsystem ID). Include the same version of all the packages from the xxxProductionProject. Add the xxxSysTestProject to your workspace.
 - Version the system test project (xxxSysTestProject) and then open a new edition of it.
 - Open new editions of the packages.
 - Determine the list of members that have been added/changed in system test by eliminating the members that match production based on the date/time stamp.
 - Export an external source format file that contains only the members that actually changed between the system test and production versions of the SUBSYS1 MSL, a second external source format file for the changes between the system test and production versions of the COMMON MSL, and a third external source format file for the changes between the system test and production versions of the DBA MSL.
 - Use **VAGen Import** to import the external source format files and create new editions of the changed parts. Select the **Defined package** radio button to put the changed editions of the parts into the packages in which they are already defined.
 - Version and release the changed classes (VAGen part classes).
 - Version any packages that had new editions of classes.
 - Create a default package and a Project List Part for the system test project (xxxSysTestProject).
 - Version the project (xxxSysTestProject).
 - Generate the programs for the system test level of code. You should only need to generate programs, map groups, and tables that have changed. Then test to be sure that what you migrated matches your system test level of code.
 3. Migrate the development level MSLs by doing the following:
 - Delete the xxxSysTestProject from your workspace.
 - Create a development level project with a name that reflects that this is the staging level of code (for example, xxxDevelopmentProject, where xxx is the subsystem ID). Include the same version of all the packages from the xxxSysTestProject. Add the xxxDevelopmentProject to your workspace.
 - Version the development project (xxxDevelopmentProject) and then open a new edition of it.

- Open new editions of the packages.
 - Determine the list of members that have been added/changed in development by eliminating the members that match system test based on the date/time stamp.
 - Export an external source format file that contains only the members that actually changed between the development and system test versions of the SUBSYS1 MSL, a second external source format file for the changes between the development and system test versions of the COMMON MSL, and a third external source format file for the changes between the development and system test versions of the DBA MSL.
 - Use **VAGen Import** to import the external source format files and create new editions of the changed parts. Select the **Defined package** radio button to put the changed editions of the parts into the packages in which they are already defined.
 - Version and release the changed classes (VAGen part classes).
 - Version any packages that had new editions of classes.
 - Create a default package and a Project List Part for the development project (xxxDevelopmentProject).
 - Version the project (xxxDevelopmentProject).
 - Generate the programs for the development level of code. You should only need to generate programs, map groups, and tables that have changed. Then test to be sure that what you migrated matches your development level of code.
4. Migrate the staging MSLs by doing the following:
- Delete the xxxSysTestProject from your workspace.
 - Create a staging level project with a name that reflects that this is the staging level of code (for example, xxxStagingProject, where xxx is the subsystem ID). Include the same version of all the packages from the xxxDevelopmentProject. Add the xxxStagingProject to your workspace.
 - Version the staging project (xxxStagingProject) and then open a new edition of it.
 - Open new editions of the packages.
 - Determine the list of members that have been added/changed in the staging MSL by eliminating the members that match development based on the date/time stamp.
 - Export an external source format file that contains only the members that actually changed between the STAGING MSL and development MSLs.
 - Use **VAGen Import** to import the external source format file and create new editions of the changed parts. Select the **Defined package** radio button to put the changed editions of the parts into the packages in which they are already defined.

- Version and release the changed classes (VAGen part classes).
 - Version any packages that had new editions of classes.
 - Create a default package and a Project List Part for the staging project (xxxStagingProject).
 - Version the project (xxxStagingProject).
 - Generate the programs for the staging level of code. You should only need to generate programs, map groups, and tables that have changed. Then test to be sure that what you migrated matches your staging level of code.
5. Assign ownership of the classes, packages, and projects.
 6. Developers can then load their own work-in-progress doing the following:
 - Export an external source format file for their own read/write MSL.
 - Load the project for the staging level of code (xxxStagingProject) into their workspace.
 - Use **VAGen Import** to import the external source format files and create new editions of the changed parts. Select the **Defined package** radio button to put the changed editions of the parts into the packages in which they are already defined.

MSLs from marketing or other demonstrations

The scenario for MSLs used in demonstrations reflects the need to have snapshots of the same MSL from various stages of development.

ENDDEMO

MSL
containing
members
built by
end of
demonstration

MIDDEMO

MSL
containing
members
built by
middle of
demonstration

STRTDEMO

MSL
containing
members
at start of
demonstration

Figure 20. Sample MSLs for a Demonstration

The three MSLs are used as follows:

- STRTDEMO is the MSL at the start of the demonstration. It might contain some partially developed members, as well as a completed server program.
- MIDDEMO contains members that are added or changed during the first part of the demonstration.
- ENDDEMO represents members that are added or changed during the entire demonstration.

This technique enables the person doing the demonstration to quickly reset the demonstration to specific scenarios.

Recommendations

Using a single package

For a small demonstration, for which all the parts can be stored into a single package, consider migrating as follows:

1. Create an external source format file for each of the three MSLs.
2. Create a new project with a name that reflects that this is for a demo (for example, DemoProject).
3. Create a new package within the project.

4. Use **VAGen Import** to import the external source format file for the STRTDEMO MSL.
5. Version and release the classes (VAGen parts).
6. Version the package.
7. Version the project.
8. Open a new edition of the project.
9. Create a new edition of the package.
10. Use **VAGen Import** to import the external source format file for the MIDDEMO MSL.
11. Version and release the classes (VAGen parts).
12. Version the package.
13. Version the project.
14. Open a new edition of the project.
15. Create a new edition of the package.
16. Use **VAGen Import** to import the external source format file for the ENDDemo MSL.
17. Version and release the classes (VAGen parts).
18. Version the package.
19. Version the project.

You can now reset the demonstration to specific points by adding the correct version of the project to your workspace.

Using multiple packages

For a larger demonstration, for which the parts need to be separated into several packages, consider migrating as follows:

1. Create an external source format file for each of the three restructured MSLs.
2. Using the MSL Migration Assistance Tool load the parts from the STRTDEMO MSL, separating them into packages as needed for your demonstration. Do not be concerned about missing parts because you know that these will be developed during the demonstration. When you commit to ENVY, specify a project name that reflects that this is a demonstration project (for example, DemoProject).
3. Version and release the classes (VAGen parts).
4. Version the packages.
5. Version the project (DemoProject).
6. Open a new edition of the project.
7. Create a new edition for each of the packages.

8. Create any new packages that are needed to contain parts that are added during the first part of the demonstration.
9. Use **VAGen Import** to import the external source format file for the MIDDEMO MSL. For new parts, move each part to the corresponding package. For existing parts, use the **Defined package** radio button to put the changes into the packages in which the parts are already defined.
10. Version and release the classes (VAGen parts).
11. Version the packages.
12. Version the project (DemoProject).
13. Open a new edition of the project.
14. Create a new edition of each of the packages.
15. Create any new packages that are needed to contain parts that are added during the second part of the demonstration.
16. Use **VAGen Import** to import the external source format file for the ENDDemo MSL. For new parts, move each part to the corresponding package. For existing parts, use the **Defined package** radio button to put the changes into the packages in which the parts are already defined.
17. Version and release the classes (VAGen parts).
18. Version the packages.
19. Version the project (DemoProject).

You can now reset the demonstration to specific points by adding the correct version of the project to your workspace.

Chapter 13. Running the MSL Migration Assistance Tool on Java

The sections that follow describe tasks you need to perform during migration, including:

- Getting ready to migrate:
 - “Starting VisualAge Generator”
 - “Creating users and setting the current user” on page 116
 - “Collecting your source code” on page 117
 - “Handling code page changes” on page 118
 - “Starting the MSL Migration Assistance Tool” on page 121
 - “Building MSL directories” on page 121
 - “Resetting the sandbox from ENVY” on page 123
 - “Selecting your MSLs” on page 125
- Moving parts to the sandbox and working with the sandbox:
 - “Selecting and migrating VAGen parts” on page 126
 - “Creating a new package” on page 129
 - “Moving a VAGen part between packages” on page 129
 - “Controlling the creation of package.nodes” on page 130
 - “Renaming a package” on page 131
 - “Collapsing a package” on page 132
 - “Handling Duplicates” on page 132
 - “Finding the package in which a part is located” on page 137
 - “Listing missing (not found) parts” on page 138
 - “Handling missing (not found) parts” on page 139
 - “Checking relationships among packages” on page 140
 - “Updating the list of required packages” on page 142
 - “Deleting a package.node” on page 143
 - “Deleting a package” on page 144
- “Committing to ENVY” on page 145

Note: ENVY provides a variety of ways of doing most tasks and supports multiple development scenarios. This chapter is intended to provide a way, but not necessarily the only way, of performing tasks related to migrating MSLs to ENVY.

Starting VisualAge Generator

This section describes how to start VisualAge Generator Version 4.0 or later, create user IDs, set the user ID for the current user, and then import and load the MSL Migration Assistance Tool.

Action	Description
Start VisualAge Generator using the VAGen Developer 4.0 on Java with Migration icon. Alternatively, from a Windows NT command prompt, change to the directory where VisualAge Generator Version 4.0 is installed and type: <code>ide /vgmig</code>	The Log window is displayed, and then the Selection Required window is displayed, prompting for the name of the user who owns this workspace.
Select Administrator from the list of users.	This enables you to define other users before running the MSL Migration Assistance Tool.
When the message saying "You must connect the repository to the current library" is displayed, select OK .	
When the message saying "Done connecting repository to the library" is displayed, select OK .	The VisualAge for Java Workbench window is displayed.

Creating users and setting the current user

You must be the Administrator to create new users.

Action	Description
From the VisualAge on Java Workbench window, select File and then Quick Start .	The Quick Start window is displayed.
Select Team Development , then Administer Users and then OK .	The User Administration window is displayed, listing the available users. The <i>Current user</i> is displayed on the lower left side above the push buttons.
Select New .	The Users Dialog window is displayed.
Specify the following:	The User Administration window is displayed with the new user added.
Unique Name	This could be the user's logon ID or employee number.
Full Name	This is the first and last name for the person.
Network Login Name	This is the user's network logon ID.
Select OK .	
Repeat the above steps to define all of your developers.	
From the Workbench window, select Workspace and then Change Workspace Owner .	A window appears prompting you to select the user.

Action	Description
Select the user that corresponds to the team leader and then select OK .	<p>The VisualAge for Java Workbench window is displayed and indicates in the title bar that the team leader is now the owner of the workspace.</p> <p>Setting the user ID for the workspace to the team leader means that the team leader becomes the class owner and package owner for all the views, VAGen part classes, and packages that are created with the MSL Migration Assistance Tool. After migration, the team leader can assign ownership to the individual developers if needed.</p>

Collecting your source code

You need to collect your source code before you can use the MSL Migration Assistance Tool. The steps necessary to do this vary depending on the environment from which you are migrating.

From Cross System Product

If you are migrating from Cross System Product, you must create an external source format file for each of your MSLs and then use the MSL Migration Assistance Tool to create an MSL directory structure. Before downloading all your external source format files, be sure to test your download program to ensure that special characters are converted correctly. You might need to define a conversion table for the download program. Also review “Using the HPTRULES.NLS file” on page 118 for information on handling the not sign (-).

From VisualAge Generator with TeamConnection and no MSLs

If you are migrating from a previous release of VisualAge Generator and use TeamConnection, you might not have MSLs. In this case you must create external source format files for your parts in TeamConnection and then use the MSL Migration Assistance Tool to create an MSL directory structure. You might want to create an external source format file for each component to help you in preserving your existing organizational structure.

In addition, if you are changing from developing on OS/2 to developing on Windows NT, be sure to review “Using the HPTRULES.NLS file” on page 118 and “Changing from OS/2 to Windows NT” on page 119.

From VisualAge Generator MSLs

If you already have VisualAge Generator MSLs and have been using the OS/2 development platform, you must migrate to the Windows NT development

platform. Be sure to review “Using the HPTRULES.NLS file” and “Changing from OS/2 to Windows NT” on page 119.

Handling code page changes

If you are migrating from Cross System Product, the code page on the host is EBCDIC and the code page on the workstation is ASCII. You should review “Using the HPTRULES.NLS file”.

If you are migrating from VisualAge Generator, the code pages for SBCS languages are different between OS/2 and Windows NT. You should review both “Using the HPTRULES.NLS file” and “Changing from OS/2 to Windows NT” on page 119.

If you are migrating from VisualAge Generator and use a DBCS language, you can skip this section because the code pages are the same for OS/2 and Windows NT.

Using the HPTRULES.NLS file

The not sign used for Cross System Product is the \neg . However, some download programs convert \neg to $\frac{1}{4}$. For VisualAge Generator, the code point for \neg differs between the OS/2 and Windows NT code pages.

The *hptrules.nls* file in the `\IBMVJava\IDE\program\hptvgj40\nls` directory enables you to specify national language information and can help with handling the not sign. One section of the file enables you to enter your three national language characters, the $\hat{\neg}$ for the not sign, and an alternate not sign. If you determine that the only special character that you need to convert is the \neg or $\frac{1}{4}$, you can do the following to handle the conversion:

1. Shut down VisualAge Generator Developer.
2. Edit *hptrules.nls*
3. Read the comments in the file. The section you need to change is the first section of the file, called `:nlsrules`.
4. In the `:nlsrules` section, there are three columns: the locale, 5 special characters, and the default language code.

The five special characters are:

Column	Description
1 - 3	The three national language characters (<code>\$#@</code> for English-US).
4	The $\hat{\neg}$ which is the standard not sign for VisualAge Generator.

- 5 An alternate not sign, which you can set to \neg , $\frac{1}{2}$, or whatever character your download program turned your not sign into.

5. Bring VisualAge Generator Developer back up.

When you use the **ESF to MSL** function (described in “Building MSL directories” on page 300), the MSL Migration Assistance Tool converts any occurrence of the alternate not sign specified in *hptrules.nls* to the $\hat{\cdot}$. This conversion only applies to occurrences in processes, statement groups, or functions. The constant delimiter for a map is not converted, but because the constant delimiter is stored with each map, conversion is not necessary. Conversion of the alternate not sign also happens when you use **VAGen Import**. Conversion does not occur when you save changes to functions. Therefore, you should use the $\hat{\cdot}$ for any new development work.

If your only code page conversion problems (whether from downloading Cross System Product code or from moving external source format files from OS/2 to Windows NT) are due to your not sign, you should be able to manipulate the *hptrules.nls* file to handle the conversion for you.

Changing from OS/2 to Windows NT

Note: If you are migrating from Cross System Product, or if you are migrating from VisualAge Generator and use a DBCS code page, skip this section. Code page conversion is not required in these situations.

The code pages from some languages differ between OS/2 and Windows NT. For example, the code point for the \neg is different for the English-US code pages for OS/2 and Windows NT. Therefore, you must convert your VisualAge Generator code if you are moving from the OS/2 development environment to the Windows NT environment. To convert between the code pages, do the following:

1. If you are migrating from VisualAge Generator 3.0 or 3.1 on OS/2, create an external source format file for each ENVY application. When migrating from any other VisualAge Generator release on OS/2, create an external source format file for each MSL. However, because GUIs cannot be migrated to Java, do not export the GUIs to the external source format file for VisualAge Generator 2.x or earlier.
2. Make the external source format file available to Windows NT.
3. From Windows NT, change to the directory in which *hptcnvXY.exe* is located. (Substitute the number of your VisualAge Generator version for the X and your release for the Y.) If you used the default directories when you installed, the directory will be `c:\IBMVJava\IDE\program`.
4. Convert the external source format file by running the following:

```
hptcnvXY esf-file-name conversion-table
```

where:

- X** Is the number of your version of VisualAge Generator — for example, use a 4 for VisualAge Generator 4.0
- Y** Is the number of your release of VisualAge Generator — for example, use a 0 (hptcnv40) for VisualAge Generator 4.0.
- esf-file-name** Is the drive, path, and file name for the external source format file you want to convert.

conversion-table

Is the name of a conversion table that translates from the OS/2 code page to the Windows NT code page.

The converted external source format file is stored as `esf-file-name.cnv` in the directory in which `hptcnvXY.exe` is located.

Use

```
hptcnvXY ?
```

(substituting your version and release numbers for **X** and **Y**) to see a short (English) description of the conversion tool.

5. Use a comparison tool to do a byte-by-byte comparison of the external source format file and the converted file.

If the only character being converted is the not sign, you might be able to use the technique described in “Using the HPTRULES.NLS file” on page 297 to avoid converting all your files. However, use caution if you skip the conversion step — some of your other files might have characters other than the not sign that require conversion.

Be especially alert for any characters being converted that should not be. For example, if you have coded:

```
MOVEA "special-character" TO HIGH-VALUES-CHAR;
```

as a method of setting the high end of a range of keys that you are searching in a file or database, you might not want the `special-character` to be converted or you might not like the hex value that results from the conversion. If this is the case, you will need to modify your code. A better technique would be to code:

```
MOVEA "special-hex-characters" TO HIGH-VALUES-HEX;
```

This technique would keep the same `special-hex-characters` in the external source format and converted files.

6. Use the converted external source format file as input to the MSL Migration Assistance Tool or **VAGen Import**.

Notes:

1. hptcnvXY does not support binary GUI tags.
2. Do not use hptcnvXY if you transferred your external source format file to Windows NT in such a way that the code page conversion was performed. For example, if you transferred the file using *ftp* with the *ascii* option, then the code page conversion should have already been done. You need to check that special characters were converted correctly.
3. Refer to the *VisualAge Generator Client/Server Communications Guide* for information about the conversion tables.

Starting the MSL Migration Assistance Tool

You can run the MSL Migration Assistance Tool on either OS/2 or Windows NT.

Action	Description
From the VisualAge on Java Workbench window, select Workspace and then Open VAGen Parts Browser .	The VAGen Parts Browser window is displayed.
From the VAGen Parts Browser window, select Tools->Migration->MSL Migration .	The MSL Migration Part List window is displayed.

Building MSL directories

After you have collected your external source format files as described in “Collecting your source code” on page 117 and handled any code page issues as described in “Handling code page changes” on page 118, you are ready to create MSL directories. The **ESF to MSL** push button on the MSL Migration Part List window enables you to build an MSL directory structure from an external source format file.

Note: Be sure that the drive where you plan to create the MSL directory supports long names. If this is not done, you might receive a return code 13 when trying to create the MSLs.

Action	Description
From the MSL Migration Part List window, select the ESF to MSL push button.	A window is displayed prompting you for the name of your external source format file.
Select the drive, path, and file name of the external source format file from which you want to build an MSL directory structure.	An Information Required window is displayed prompting you for the name of a directory into which the MSL Migration Assistance Tool can build the MSL.

Action	Description
Enter the drive and path for the directory and select the OK push button.	<p>If the directory you specified exists, a message is displayed prompting you to confirm the directory name.</p> <p>If the directory does not exist, a message is displayed asking if you want the MSL Migration Assistance Tool to create the directory.</p> <p>After you have responded Yes to either message, an Operation in Progress message, an Operation in Progress window is displayed. When the window disappears, the parts (members) from the external source format file have been created in the MSL directory that you specified. All the actions in “Automatic conversions during 4.0 migration” on page 48 have been performed.</p>
If there were any parts that contained errors in the external source format files, a List of parts that could not be read window is displayed followed by the VAGen Parser Messages window.	<p>The list contains the names of the parts that were not written to the MSL directory. You can write this list to the mslmig.log file for future reference.</p> <p>The VAGen Parser Messages window contains both information and error messages. Parts that had no messages or only information messages are already in the MSL directory. You should ignore the information messages. Examples of the information messages include:</p> <pre data-bbox="801 1065 1206 1298"> HPT.PE.290.i The CICS/OS attribute value was replaced with the MVS CICS attribute value HPT.PE.17.i The transaction name on the SEGTRAN attribute is not valid HPT.PE.21.i The FILETYPE VSAMCICS is no longer supported. It was changed to VSAM.</pre>
Parts with error messages were not written to the MSL directory. You need to correct these errors by:	<ul style="list-style-type: none"> • Correcting the part in Cross System Product or VisualAge Generator and then exporting the external source format file for the part again and using the ESF to MSL push button to add the part to the MSL directory. • Correcting the external source format file and using the ESF to MSL push button to process the external source format file again.

Resetting the sandbox from ENVY

If you have previously migrated a subsystem and are now ready to do another subsystem, you might need to reset the MSL Migration Assistance Tool to reflect the parts that are currently in ENVY. This can happen if you migrate one subsystem, work with it for a while in ENVY, and then decide to migrate your remaining subsystems. Particularly if the subsystems to be migrated now share code with the subsystem that was previously migrated, you should reset the MSL Migration Assistance Tool to reflect the current code in ENVY.

Note: You cannot load a package into the sandbox from your ENVY workspace in the following situations:

- You added a nonvisual part or a Java class.
- You added parts for VisualAge Generator control information (generation options, linkage table, resource associations, bind control commands, or linkage editor control statements).

However, you can unload these classes so that the package can be reloaded into the sandbox.

Action	Description
From the VisualAge for Java Workbench window, add into your workspace any packages (such as your common packages) that you need to have in the sandbox when you migrate your new MSLs.	The packages you need should be displayed in the VisualAge on Java Workbench window.
Alternatively, from the Projects Browser window, add the projects that contain the packages you need in your workspace.	

Action	Description
From the MSL Migration Part List window, select the ENVY Pkg Selection push button.	<p>If there are any packages in the sandbox, a message is displayed asking if they can be deleted.</p> <p>If you respond Yes to the message about deleting existing packages or if there were no packages, a Selection Required window is displayed, prompting you to specify the packages already in ENVY that you want to load into the sandbox. Only packages in your workspace can be loaded into the sandbox.</p> <p>If you respond No, the list of existing packages in the sandbox is not cleared, and your new selections are added to the existing sandbox list. A Selection Required window is displayed, prompting you to specify the packages already in ENVY that you want to load into the sandbox. Only packages in your workspace can be loaded into the sandbox.</p> <p>If you respond Cancel, the sandbox is not reset from ENVY.</p>
Select one or more packages from the left pane and then select >> to move them to the right pane.	The packages you selected are listed in the right pane.
Select the OK push button.	<p>The packages and the parts they contain are loaded into the sandbox and marked as <i>Committed</i>.</p> <ul style="list-style-type: none"> • There is no analysis of the parts to determine if there were missing (<i>Not Found</i>) parts. • The timestamp of the part in the sandbox is the timestamp from the edition currently loaded in your workspace, not the timestamp from the original MSL.
You must specify your MSL library selections again (see “Selecting your MSLs” on page 125).	

Action

Description

If you receive a message saying “Unsuccessful in reading ENVY pkg into tool”, it is because you have added a nonvisual part, a Smalltalk class, or a VisualAge Generator control information part (generation options, linkage table, resource associations, bind control commands, or linkage editor control statements) to the package. The migration log file (**mslmig.log**) lists the parts that prevent the package from being loaded into the sandbox. Do the following so that you can load the package into the sandbox:

- If the package and the class are already versioned:
 - Open a new edition for the package.
 - Delete the classes that cause a problem.
- If the package and the class are not versioned yet:
 - Version the classes that cause a problem.
 - Delete the classes that cause a problem.

Then reload the package into the sandbox.

After you have reloaded the package into the sandbox, add those classes back into your workspace now. This ensures that when you version and release classes after committing new or changed packages to ENVY all the classes for this package will be there.

Selecting your MSLs

The MSL Migration Assistance Tool works from an MSL concatenation sequence to move parts into the sandbox. You specify your MSLs and the concatenation sequence as follows:

Action

Description

From the MSL Migration Part List window, select the **MSL Library Selection** push button.

The MSL Library Selection window is displayed.

In the *Basic MSL directory*, enter the drive and directory for a basic MSL that you want to process. For example:

```
f:\ms1s\mys1
```

The directory you specified is added to the *MSL concatenation* area at the bottom of the MSL Library Selection window.

The basic MSL directory that you specify can be an MSL from a previous release of VisualAge Generator or an MSL that you created using the technique described in “Building MSL directories” on page 121.

Select the **Add** push button.

Action	Description
Repeat the previous step until you have all your basic MSLs defined.	The MSL Library Selection window lists all your basic MSLs. The order in which they are listed is the order in which they will be searched for parts.
If you need to change the MSL concatenation sequence, select one MSL in the <i>MSL concatenation</i> area.	The <i>MSL concatenation</i> area changes to reflect the new MSL concatenation sequence.
Then select one of the following to change the concatenation order:	Repeat this step until you are satisfied with the concatenation sequence.
Move Up To move the selected MSL one position higher (earlier) in the concatenation sequence.	
Move Down To move the selected MSL one position lower (later) in the concatenation sequence.	
Remove To delete the selected MSL from the concatenation sequence. The MSL directory is not deleted; only the concatenation sequence is affected.	
When you have the MSL directories listed in proper order for your concatenation sequence, select OK .	The Part List Selection Criteria View window is displayed, with the MSL directories you specified listed under <i>Libraries</i> . The VG Part Prerequisites View window is also displayed.

Selecting and migrating VAGen parts

To migrate a VAGen part and its associates to an ENVY package, you first select the part and then indicate whether you want it to be placed in an existing ENVY package or in a new package. You can work with a group of VAGen parts or a single VAGen part at a time.

This step only moves the parts to the sandbox; they are not moved to the ENVY repository until you commit the packages to ENVY. It is easier to change the organization of your parts while they are in the sandbox than after they are in the ENVY repository.

Action	Description
<p>From the Part List Selection Criteria View window:</p> <ul style="list-style-type: none"> • Select the VAGen part type(s) you want to process. Use the All Types push button to select all the part types. Use the Reset push button to deselect all the part types. If you are migrating from existing MSLs, select the Process and/or Statement Group part types. If you are migrating from pseudo MSLs created from external source format files, select the Function part type. • Use a wildcard in the Part name field to limit the search. • Select the Only parts not processed toggle button to limit the list of VAGen parts to those that have not been processed by the MSL Migration Assistance Tool. Deselect the Only parts not processed toggle button to include all the parts that satisfy the part type and wildcard search criteria. • Select the Build List push button. 	<p>The MSL Migration Part List window is displayed with the selected VAGen parts.</p> <p>The Status column is set to <i>Processed</i> for any parts that have already been moved to the sandbox.</p>
<p>Note: Unlike the VisualAge Generator Member Selection List, all libraries listed in the <i>Libraries</i> area of the Part List Selection Criteria View window are searched for parts. You cannot limit the search to certain MSLs by highlighting the directories listed in the <i>Libraries</i> area.</p>	
<p>To help locate parts within the MSL Migration Part List window, you can sort the parts by selecting VAGen Parts and then selecting one of the following:</p> <ul style="list-style-type: none"> • Sort by Type • Sort by Name • Sort by Library 	<p>The order of the parts is changed to match your specified sort criteria.</p>

Action	Description
<p>From the MSL Migration Part List window, select the VAGen part(s) that you want to process with the MSL Migration Assistance Tool. The selected VAGen part and any of its associates that have not yet been migrated will be processed together.</p>	<p>The selected parts are highlighted. If you use Find, Select All, or Select Parts Not Processed any previously selected parts are deselected.</p>
<p>To select multiple parts, hold down the Ctrl while you select each of the parts you want to process.</p>	
<p>You can also select VAGen Parts and then one of the following to help in selecting parts:</p>	
<ul style="list-style-type: none"> • Find to select one or more parts that satisfy your selection criteria. You can use a wildcard when you specify the selection criteria. For example, if you specify <i>V*</i>, all parts that begin with <i>V</i> are selected. If you specify only a <i>V</i>, the part named <i>V</i> (if it exists) is selected. • Select All to select all parts on the list. • Deselect All to deselect all parts on the list. • Select Parts Not Processed to select only those parts that are not yet in the sandbox. • Select Not Found Parts to select only those parts that are listed as Not-Found in the Status column. • Select Duplicate Parts to select only those parts that are listed in the Duplicate column. 	
<p>Select VAGen Parts, then Selected and then one of the following:</p>	<p>The VG Part Prerequisites View window is displayed. You can then look at the packages that have been changed and see the parts in them.</p>
<ul style="list-style-type: none"> • Create Single Package to place all of the selected VAGen parts and their associates into the same new package. An Information Required window is displayed prompting you for the name of the new package. • Create Multiple Packages to place each of the selected programs with its associates into a separate package. Each package will be named <i>xxxxx.pkg</i> where <i>xxxxx</i> is the name of the corresponding program. • Add into Package to place all of the selected VAGen parts and their associates into the same package that already exists in the sandbox. In the Selection Required window that is displayed, select the package into which these parts will be added. 	<p>If any associates of the parts that were just moved to the sandbox were already in the sandbox, one or more <i>package.nodes</i> might be created. A <i>package.node</i> is created for each part or parts that are shared by two explodable packages.</p>

Creating a new package

If you add VAGen parts with their associates to an ENVY package, and then review the list of VAGen parts that are now in the ENVY package, you might find that there are some parts (for example, common code) that would be better placed in a separate package, but this separate package does not yet exist.

This section describes how to create a new ENVY package using the MSL Migration Assistance Tool.

Action	Description
From the VG Part Prerequisites View window, select Packages and then Create Package .	An Information Required window is displayed, prompting you for the package name.
Type the name of the ENVY package.	The VG Part Prerequisites View window is refreshed and includes the name of the new package. See “Moving a VAGen part between packages” for information about moving VAGen parts from another package into this new package.

Moving a VAGen part between packages

After adding VAGen parts to a package, particularly after adding a program and its associates to a package, you should review the list of VAGen parts for the package in the **VG Part Prerequisites View** window to determine whether any VAGen parts have been included that would be better placed in a different package. For example, some common code might have been included in the associates list for a program being moved. Reviewing the list of VAGen parts for the package helps to find common code that is better placed in a separate package. If you used naming conventions to distinguish common code, these common VAGen parts are fairly easy to find in the **VAGen Parts** pane of the VG Part Prerequisites View window.

This section describes the steps to move a VAGen part from one package (FromPackage) to another (ToPackage) using the VG Part Prerequisites View window.

Note: After you commit a package to ENVY, you can no longer move its parts to a different package using the MSL Migration Assistance Tool. Refer to the *VisualAge for Java Getting Started* document for information on moving parts between packages in ENVY.

Action	Description
<p>From the VG Part Prerequisites View window:</p> <ul style="list-style-type: none"> • In the Packages pane, select the FromPackage — the package from which you want to move VAGen parts. • In the VAGen Parts pane, select the VAGen parts you want to move. <p>You can select VAGen Parts and then select one of the following to help in selecting parts:</p> <ul style="list-style-type: none"> – Find Parts to select one or more parts that satisfy your selection criteria. You can use a wildcard when you specify the selection criteria. For example, if you specify V*, all parts that begin with V are selected. If you specify only a V, the part named V (if it exists) is selected. When you use Find Parts, any previously selected parts are deselected. – Sort Parts by Type to group parts based on the part type. Any previously selected parts are still selected. – Sort Parts by Name to sort the parts based on the part name. Any previously selected parts are still selected. 	<p>The Selection Required window is displayed, listing the other packages.</p>
<p>Select VAGen Parts and then Move or Move with associates.</p>	<p>The VG Part Prerequisites View window is displayed, showing the FromPackage without the VAGen parts you moved.</p>
<p>Select the ToPackage.</p>	<p>If you selected Move, only the selected parts were moved.</p>
<p>Select the OK push button.</p>	<p>If you selected Move with associates, the selected parts and their associates in the FromPackage are moved to the ToPackage.</p>

You might need to update the prerequisites for the FromPackage to include the ToPackage.

See “Checking consistency of packages” on page 141 for information about doing a consistency check to determine which prerequisites need to be changed. You should do a consistency check on the FromPackage and all its dependent packages.

Controlling the creation of package.nodes

Some parts are used by many programs and views. For example, records, tables, and data items are typically shared by multiple programs and views. Similarly functions (processes and statement groups) might also be shared.

You might need to have programs and views that share common parts in different ENVY packages. A common part can be stored in one of the ENVY packages that uses the part or placed in an ENVY package that contains just common parts.

When you migrate a part, its associates are considered at the same time. An associate that is not yet in the sandbox is placed in the same ENVY package. An associate that is already in the sandbox is treated differently based on the following:

- If the package containing the associate is marked *explodable*, the associate is moved to a new *package.node*. The original package and the package being created for the part now being moved to the sandbox specify the new *package.node* as a prerequisite.
- If the package containing the associate is marked *unexplodable*, the associate is not moved to a new *package.node*. The package for the part now being moved to the sandbox adds the package containing the associate as a prerequisite.

You can change whether a package is *explodable* or *unexplodable* by changing it in the VG Part Prerequisites View window.

Note: After a package is committed to ENVY, it is *unexplodable*.

Action	Description
<p>To change all packages in the sandbox, select Packages and then one of the following:</p> <ul style="list-style-type: none"> • Set All Unexplodable to mark all the packages in the sandbox as <i>unexplodable</i>. • Set All Explodable to mark all the packages in the sandbox as <i>explodable</i>. 	<p>The VG Part Prerequisites View window is displayed.</p> <p><i>Unexplodable</i> packages are prefixed with an asterisk (*). The * is not part of the package name.</p> <p><i>Explodable</i> packages do not have a prefix.</p>
<p>To change a single package in the sandbox, select the package, then select Packages, then Selected and then one of the following:</p> <ul style="list-style-type: none"> • Toggle Explode to change the current setting for the selected package. • Set Unexplodable to mark the selected package as <i>unexplodable</i>. • Set Explodable to mark the selected package as <i>explodable</i>. 	<p>The VG Part Prerequisites View window is displayed.</p> <p><i>Unexplodable</i> packages are prefixed with an asterisk (*). The * is not part of the package name.</p> <p><i>Explodable</i> packages do not have a prefix.</p>

Renaming a package

You might want to rename a package in the following situations:

- All the parts in a *package.node* belong together and should become a package with a more meaningful name.
- The name you originally chose for the package would be more meaningful if it was changed.
- You typed the package name incorrectly when you created the package.

Action	Description
From the VG Part Prerequisites View window, select the package you want to rename, then select Packages and then Selected → Rename .	An Information Required window is displayed prompting you for the new package name.
Type the new name for the package.	The VG Part Prerequisites View window is displayed and shows the new package name. Packages that specified the renamed package as a prerequisite have been updated to reflect the new name.

Collapsing a package

You might want to merge one package into another package. This might occur if you decide that the two packages will be maintained by the same developer or that they share so many parts that it would be better to combine them. The MSL Migration Assistance Tool calls this process *collapsing*.

Action	Description
From the VG Part Prerequisites View window, select the package you want to collapse, then select Packages and then Selected → Collapse .	An Information Required window is displayed with a combo box listing the packages that exist in the sandbox.
Select the name of the package into which you want to merge and then select the OK push button.	The VG Part Prerequisites View window is displayed. The package that you collapsed no longer appears. If you select the package into which you merged, the parts displayed include the parts from the package you collapsed. Any package that specified the collapsed package as a required package has been updated to reflect the name of the package into which you merged.

Handling Duplicates

You might have duplicate members. This can occur due to:

- Controlled duplicates such as a message table that differs between subsystems as described in “Multiple subsystems with controlled duplicates” on page 89.
- Unintended duplicates as described in “MSLs that contain unintended duplicates” on page 97.
- Duplicates due to using templates and then using a separate MSL for members that were modified due to business logic as described in “MSLs containing code from VisualAge Generator Templates or BW*Wizard” on page 278.

The MSL Migration Assistance Tool helps identify the duplicates and enables you to select the version that you want to migrate. However, it cannot determine for you which level of code is the current version.

Within a single MSL concatenation sequence, to find duplicates easily in the MSL Migration Part List window, select **VAGen Parts** and then **Sort by Name**.

Controlled Duplicates

When you have controlled (intended) duplicates between subsystems, do the following after committing the first subsystem to ENVY, but before starting to migrate the second subsystem.

Action	Description
From the VisualAge on Java Workbench window, delete the project and packages for the first subsystem from your workspace. Do not delete the project(s) for common code.	The deleted project and packages should no longer appear in the VisualAge on Java Workbench window.
From the VG Part Prerequisites View window, delete all the packages for the first subsystem from the sandbox. See “Deleting a package” on page 144 for information on how to do this.	Only the packages created for the common code that is shared by the subsystems should be listed in the VG Part Prerequisites View window.
If you are not able to delete all the packages for the first subsystem from the VG Part Prerequisites View window, select Packages and then Delete All .	All of the packages are deleted from the VG Part Prerequisites View window. Now you can reload the common packages from ENVY as described in “Resetting the sandbox from ENVY” on page 123.

Unintended Duplicates

When you have unintended duplicates, you need to determine which version of the part is the one that should be committed to ENVY. These parts will have *True* in the **Duplicate** column. The options for handling duplicates are:

- Remove Duplicate
- Replace Existing

If you remove a duplicate part:

- The version of the part is removed from future consideration and will no longer appear on the MSL Migration Part List window, even if you specify *Only parts not processed*.
- The *removed duplicate* part is ignored for any other processing. For example, if the part is an associate of something moving to the sandbox, the first found version that has not been removed, is what will be moved to the sandbox.

If you replace an existing part in the sandbox:

- The part and any associates that are not currently in the sandbox are moved to the sandbox. The part is moved to the same package where the existing part is already located. The associates are handled as follows:
 - If the associate is already in the sandbox, then
 - If the associate is in the same package as the part, nothing happens.
 - If the associate is in a different package from the part, and is in an unexplodable package, nothing happens.
 - If the associate is in a different package from the part, and is in an explodable package, a package.node is created.
 - If the associate is not yet in the sandbox, then:
 - If the associate is not a duplicate (or you have previously removed duplicates so this is the only version of the associate still in consideration) in the current MSL concatenation sequence, it is moved to the sandbox in the same package as the part that is being replaced.
 - If the associate is a duplicate in the current MSL concatenation sequence, the first found version from the concatenation sequence is moved to the sandbox in the same package as the part that is being moved. If you display the MSL Migration Part List window, the duplicates will be listed and you can then Replace Existing if you want something other than the first found associate to go into the sandbox.
 - All parts with the same name in the MSL Migration Part List window are updated so their **Status** and **Last Migration Library Timestamp** reflect the part that is now in the sandbox. In addition, any associated parts that are moved to the sandbox are updated on the MSL Migration Part List.

Which options are available for a particular duplicate part depends on the version of the part listed on the MSL Migration Part List and on the version in the sandbox.

- If the **Last Migration Library Timestamp** is filled in on the MSL Migration Part List window, you can:
 - Replace duplicate (put this version of the part into the sandbox, replacing the part that is already there)

- Remove duplicate (remove this version from future consideration)
- If the **Last Migration Library Timestamp** is blank, what you can do depends on whether this version of the part is from the same MSL as what is in the sandbox:
 - If the **Status** is *Not Found*, you can remove the duplicate, but you cannot replace the existing part. This situation occurs when you have a part that was previously identified as *Not Found*, but you have now found two or more versions of it in your current MSL concatenation sequence. After you have removed duplicate versions of the *Not Found* part, you can use **Add Not Found Part** to update the sandbox (see “Handling missing (not found) parts” on page 139).
 - If this version is from the same MSL, you cannot remove the duplicate or replace the existing part.
 - If this version is from a different MSL, you can remove the duplicate, but you cannot replace the existing part. A blank **Last Migration Library Timestamp** in this situation indicates that this version of the part has the same timestamp as the part in the sandbox and is therefore the same version, but from a different MSL.

Action	Description
<p>To remove a duplicate from further consideration:</p> <ul style="list-style-type: none"> • From the MSL Migration Part List window, select the version of the part that you do not want to move to the sandbox. • Select VAGen Parts and then Selected → Handle Duplicate Parts → Remove Duplicate. 	<p>The MSL Migration Part List window is updated and this version of the part is no longer included in the list.</p>

Action	Description
<p>To replace an existing part in the sandbox:</p> <ul style="list-style-type: none"> From the MSL Migration Part List window, select the version of the part that you want to use to update the sandbox. Select VAGen Parts and then Selected → Handle Duplicate Parts → Replace Existing. 	<p>The MSL Migration Part List window is updated and the Status and Last Migration Library Timestamp columns are updated.</p> <p>The part is updated in the sandbox. Any associated parts that were not yet in the sandbox have now been moved to the sandbox.</p> <p>If you replaced a part in a package that has already been committed to ENVY:</p> <ul style="list-style-type: none"> The package name in the VG Part Prerequisites View window is changed to indicate that it has been <i>Modified</i> and that a consistency check must be performed. The need for a consistency check is indicated by a tilde (~) to the right of the package name. See “Checking consistency of packages” on page 141 for information on how to check that all required packages are specified. After you have done the required consistency check, you can commit the <i>Modified</i> packages to ENVY. This creates a new edition of the duplicate parts in ENVY. The part names of the duplicate parts also have a tilde beside them to indicate that they have been modified.

Duplicates for business logic

Duplicates occur for business logic when you have built a program from templates, imported it into one MSL, and then made changes to some members in another MSL. These duplicates appear during migration after you have migrated and committed the template-built parts from their MSL and then change to the MSL that contains the business logic. When you try to migrate the business logic MSL, there will be two types of parts:

- Parts that were not originally built by the templates, but which you added for business logic. These parts will have blanks in the **Duplicate** and **Last Migration Library Timestamp** columns in the MSL Migration Part List window. You handle them like any other parts. In most cases, you will want to add them into an existing (committed) package. The MSL Migration Assistance Tool will mark the package as having been *Modified* and you will be able to commit the package to ENVY again to put the new parts into the ENVY repository.

- Parts that were originally built for the templates that you modified for business logic. These parts will have *True* in the **Duplicate** column and the timestamp from the template-built MSL in the **Last Migration Library Timestamp** column. You handle these parts as described in the following steps.

Action	Description
Select the duplicate parts. Select VAGen Parts and then Select Duplicate Parts to select all the duplicates at one time.	Each selected part is moved to the sandbox and placed in the same package in which it already exists.
Next select VAGen Parts and then Selected → Handle Duplicate Parts → Replace Existing .	<p>The package name in the VG Part Prerequisites View window is changed to indicate that it has been <i>Modified</i> and that a consistency check must be performed. The need for a consistency check is indicated by a tilde (~) to the right of the package name. See “Checking consistency of packages” on page 141 for information on how to check that all required packages are specified. After you have done the required consistency check you will be able to commit the <i>Modified</i> packages to ENVY. This creates a new edition of the duplicate parts in ENVY</p> <p>The part names of the duplicate parts also have a tilde beside them to indicate that they have been modified.</p>

Finding the package in which a part is located

You might need to determine which package a part or group of parts is currently located in. For example, you might have some common code for which the part names all start with XYZ and you want to verify that you have placed all the parts that begin with XYZ into the same package.

Action	Description
From the VG Part Prerequisites View window, select Packages and then Find Parts .	An Information Required window is displayed.

Action

You can use a wildcard when you specify the selection criteria. For example, if you specify *V**, all parts that begin with *V* are selected. If you specify only a *V*, the part named *V* (if it exists) is selected. When you use **Find Parts**, any previously selected parts are deselected.

Description

If parts satisfying the selection criteria are located in several packages, a Selection Required window is displayed and you can select the package you want to review.

If parts satisfying the selection criteria are located in only one package, the VG Part Prerequisites View window is displayed with this package highlighted and all the parts within this package that satisfy the selection criteria also highlighted.

Listing missing (not found) parts

When you move parts to the sandbox, their associates generally move with them. However, in some cases the associates might not exist in your MSL concatenation sequence. This can occur for the following reasons:

- Some parts have not yet been developed for a new project.
- Some parts are intentional duplicates that exist in MSLs that will be processed later. For example, see “Multiple subsystems with controlled duplicates” on page 89.
- Some parts have been lost over time or possibly are in MSLs that you have not considered for migration.

You need to resolve these associates that cannot be found — either determine that there is not a problem or locate the missing code. The steps described in the following table enable you to obtain a list of the missing parts. However, you must handle the resolution.

Action

From the VG Part Prerequisites View window, you can list missing parts for all packages in the sandbox, by selecting **Packages** and then **List All Not Found Parts**.

Description

A **List of not-found parts** is displayed. It shows all parts that should have been moved to the sandbox but for which source could not be found in the MSL concatenation sequence. The package name that expected to contain the missing part is also displayed.

If all associated parts are found in the sandbox, an Information window is displayed, indicating that there are no *not found* parts.

Action	Description
Select Write To File to put a copy of this list in the migration log file, mslmig.log .	<p>If you scroll to the bottom of the migration log file, it displays the list of parts that could not be found.</p> <p>You can print the migration log file if you need a hardcopy to help in finding the parts.</p>
From the VG Part Prerequisites View window, you can list missing parts for one package in the sandbox, by selecting the package, then select Packages and then Selected → List Not Found Parts .	<p>A List of not-found parts is displayed. It shows all parts within the selected package that should have been moved to the sandbox but for which source could not be found in the MSL concatenation sequence. The package name that expected to contain the missing part is also displayed.</p> <p>If all associated parts are found in the sandbox, an Information window is displayed indicating that there are no <i>not found</i> parts.</p>
Select Write To File to put a copy of this list in the migration log file, mslmig.log .	<p>If you scroll to the bottom of the migration log file, it displays the list of parts that could not be found.</p> <p>You can print the migration log file if you need a hardcopy to help in finding the parts.</p>

Handling missing (not found) parts

When you find the MSL that contains a part that was identified as *Not found*, you need to move the part to the sandbox. Parts in the current MSL that were previously identified as *Not found* are identified in the MSL Migration Part List window by a **Status** of *Not Found*.

Note: If you have now located multiple versions of the previously *Not Found* part, you must first remove duplicates, as described in “Unintended Duplicates” on page 133. This enables the MSL Migration Assistance Tool to know which of the versions you want to move to the sandbox.

Action	Description
<p>From the MSL Migration Part List window, select the part. If there are several <i>Not Found</i> parts, you can select them all at once by selecting VAGen Parts and then selecting Select Not Found Parts.</p> <p>Next select VAGen Parts and then Selected → Add Not Found Part.</p>	<p>The part is moved to the package in the sandbox that currently contains the part. The <i>Not Found</i> indicator to the right of the part name in the sandbox is removed.</p>

If the part was moved to the `notfound.pkg` because you had already committed packages to ENVY, you can

- Move the part to the committed package which expected to have the part. This changes the status of the committed package to *Modified* and you will need to commit the package to ENVY again.
- Move the part to a package that has not yet been committed to ENVY. This situation occurs in the scenario described in “Multiple subsystems with controlled duplicates” on page 89 for the message table part. The message table part is *Not found* when the COMMON and DBA MSLs are migrated. However, it exists in the SUBSYS1 MSL and needs to be placed into one of the packages being created for SUBSYS1 when you migrate the SUBSYS1 MSL.

This situation can also occur in the scenario described in “Separate production MSLs for each developer” on page 272. In this scenario, each subsystem must develop its own version of any previously *Not found* parts.

Checking relationships among packages

You might want to determine what parts in a package are referenced by another package or confirm that all prerequisite relationships have been established. You do this using the techniques described in the following sections:

- “Determining which programs are referenced”
- “Determining the parts that are referenced” on page 141
- “Checking consistency of packages” on page 141

Determining which programs are referenced

Called programs are not identified as associates of the functions (processes or statement groups) that call them. However, when you are testing, you need to have any called programs available. Therefore, when you define your projects, you might want to include the packages containing both the called and calling programs in the same project.

Note: Checking called programs can only be done before you commit packages to ENVY.

Action	Description
Select the package for which you want to determine the programs called by any functions (processes or statement groups) it contains.	A list of called programs is displayed.
Select Packages and then Selected → Programs Referenced .	
Select Write To File to put a copy of this list in the mslmig.log file.	

Determining the parts that are referenced

In some cases, you might want to determine which parts from one package are referenced by another. For example, if there is only one part that is used in a required package, you might want to move the part to reduce the number of required packages. This function can be done even if one of the packages involved has been committed to ENVY.

Action	Description
Method 1 - Determining which parts in a package are referenced by parts in a particular dependent package.	A list of associated parts in the selected package that are referenced by parts in the dependent package is displayed.
Select the package and then select one of its dependent packages.	The chain of required packages is not considered.
Select Dependent Packages and then Parts Referenced .	
You can save this list of referenced parts by selecting Write To File . The list is copied to the migration log file, mslmig.log .	Scroll to the bottom of the migration log file to see the list of dependent packages. You can also print the migration log file to use this list as a reference.
Method 2 - Determining which parts in a particular required package are referenced by parts in a package.	A list of associated parts in the required package (and its required packages) that are referenced by parts in the selected package is displayed.
Select the package and then select one of its required packages.	The chain of required packages is considered.
Select Required Packages and then Cross Reference .	
You can save this list of referenced parts by selecting Write To File . The list is copied to the migration log file, mslmig.log .	Scroll to the bottom of the migration log file to see the list of referenced parts. You can also print the migration log file to use this list as a reference.

Checking consistency of packages

When you move parts from one package to another in the sandbox or when you replace a duplicate part in the sandbox, the required packages might not

be updated correctly. To check that all the necessary required packages are specified, do the steps described in this section.

Note: In the VG Part Prerequisites View window, if a package name has a tilde (~) to the right of the name, its required packages might not be correct. Until you check consistency for this package, the MSL Migration Assistance Tool will not allow you to commit the package to ENVY.

Action	Description
From the VG Part Prerequisites View window, you can verify that all required packages are specified for a package, by selecting the package, then selecting Packages and then Selected → Check Consistency .	
If the associates of all parts in the selected package can be found in one of the required packages, an Information window is displayed stating <i>No inconsistency found</i> .	Select the OK push button to close the Information window.
If one or more of the associates of some parts in the selected package cannot be found in the required packages, the “List of inconsistency” window is displayed. It shows the parts that could not be referenced based on the current required packages. The package which needed to use each of the parts is also shown.	See “Finding the package in which a part is located” on page 137 for information about how to find a part in the sandbox. See “Changing the list of required packages” for information about how to change the list of required packages.
Select Write To File to put a copy of this list in the mslmig.log file.	

Updating the list of required packages

After you have checked the consistency of a package, you might need to modify its list of required packages. In addition, you might want to ensure that the list of required packages does not include any unnecessary packages. These techniques are described in:

- “Changing the list of required packages”
- “Normalizing the list of required packages” on page 143

Changing the list of required packages

Sometimes you might want to add or delete a required package from the list. For example, after running a consistency check as described in “Checking consistency of packages” on page 141, you might need to add a required package.

Action	Description
From the VG Part Prerequisites View window, select the package for which you want to change its list of required packages. Then select Required Packages and then Change .	A Selection Required window is displayed.
<p>You can:</p> <ul style="list-style-type: none"> • Add a required package by moving it from the left pane to the right pane. • Remove a required package by moving it from the right pane to the left pane. 	The VG Part Prerequisites View window is refreshed and shows the updated list of required packages.
When you are satisfied with the list of required packages, select the OK push button.	

Normalizing the list of required packages

You might want to check that unnecessary packages are not included in the list of required packages. However, when you commit a package to ENVY, the package's list of prerequisites is not used. However, you might want to record this information for use in creating PLPs.

Action	Description
From the VG Part Prerequisites View window, select the package for which you want to normalize the required packages.	The <i>Required Packages</i> pane is refreshed. Any unnecessary packages have been removed.
Then select Packages , and then Selected → Normalize Prerequisites .	

Deleting a package.node

A package.node is created when a VAGen part that is already being used by an existing package is an associate of the VAGen parts being put into another package. The new package.node reflects the VAGen parts that are common with an existing package and the existing package is not marked as *unexplodable*.

You should review the VAGen parts in the new package.node and determine where they should be placed — in the existing package, in the package to which you were moving VAGen parts, or possibly in a third package that contains common VAGen parts.

If all parts in the package.node should be moved to the same package, it is easier to collapse the package.node as described in “Collapsing a package” on page 132.

However, when parts must be moved to different packages, if you move a part with its associates, all other parts in the package.node might be associates of the part you moved. This creates a situation in which the package.node exists, but contains no parts. See “Moving a VAGen part between packages” on page 129 for information about moving a VAGen part to a different package.

This section describes how to delete a package node after all VAGen parts in it have been moved to other nodes and all dependents have been removed.

Action	Description
From the VG Part Prerequisites View window, select the package.node to be deleted.	The VAGen parts, required packages, and dependent packages for the selected package appear in the VG Part Prerequisites View window.
If there are VAGen parts, see “Moving a VAGen part between packages” on page 129 for information on how to move a VAGen part or “Collapsing a package” on page 132 for information on collapsing the entire package.node into one package.	
If there are dependents, see “Changing the list of required packages” on page 142 for information on deleting the dependents.	
After all the VAGen parts and dependents have been removed, then select Packages and then Selected → Delete .	The VG Part Prerequisites View window is refreshed and the package.node has been deleted.

Deleting a package

After you have migrated one subsystem to ENVY, you might need to delete the packages created for that subsystem from the sandbox. This occurs in the scenarios described in “Multiple subsystems with controlled duplicates” on page 89, “Separate production MSLs for each developer” on page 272 and “MSLs that contain unintended duplicates” on page 97.

Deleting one package

To delete one package, follow the steps described below.

Action	Description
From the VG Part Prerequisites View window, select the package to be deleted.	The VAGen parts, required packages and dependent packages for the selected package appear in the VG Part Prerequisites View window.

Action	Description
<p>The package you want to delete must not be specified as a required package by any other package. For example, if AppA lists AppB as a Dependent Package, AppB also lists AppA as a Required Package. You cannot delete AppA until you remove AppA from the list of required packages for AppB.</p> <p>The packages listed in the Dependent Packages pane are the packages that specify the package you want to delete as a required package. If there are dependents, see “Changing the list of required packages” on page 142 for information on deleting this package from the dependents’ list of required packages.</p>	<p>The list of Dependent packages must be empty.</p>
<p>Select Packages and then Selected → and then Delete.</p>	<p>If the package has no dependents, it is deleted.</p> <p>If it has dependents, an error message is displayed and the package is not deleted.</p>

Deleting all packages

You might want to delete all packages in the sandbox in the following situations:

- After a pilot migration, you might want to clear the sandbox without committing any packages to ENVY so that you can try a different organizational structure for your packages.
- You have finished migrating a group of packages that shared one set of common parts and you want to reset the sandbox before migrating another group of packages with a different set of common parts.

Action	Description
<p>Select Packages and then Delete All.</p>	<p>The VG Part Prerequisites View window is refreshed with all packages cleared from the window.</p>

Committing to ENVY

When you are satisfied with the packages, their VAGen parts, and their lists of required packages, you need to commit this work to the ENVY repository to save it. You can commit one package at a time or all packages at once.

Note: All packages committed at the same time must go into the same project.

Action	Description
<p>To make sure that all parts have been considered, from the Part List Selection Criteria View window, do the following:</p> <ul style="list-style-type: none"> • Select the All Types push button • Select the Only parts not processed toggle button • Select the Build List push button. 	<p>The MSL Migration Part List window is refreshed and contains any parts that have not been moved to the sandbox.</p> <p>Generally, there should not be any parts on this list. If there are, you should determine whether they represent obsolete parts or parts that should be migrated now. For example, you might have parts in your common MSL that are not referenced within the MSL itself but which will be referenced when you migrate MSLs for your subsystems. You should migrate these common parts at this time</p> <p>This helps to insure that all parts are considered by the MSL Migration Assistance Tool.</p>

From the VG Part Prerequisites View window, you should also do the following:

- Review the number of parts in each package. This number is listed to the right of the package name in the VG Part Prerequisites View window. If the number of parts of any one type is greater than 600-700, you might want to consider splitting this package into smaller packages to improve performance.
- List any parts not found (see “Listing missing (not found) parts” on page 138).
- For any package with a tilde (~) to the right of its name, check that the required packages are specified (see “Checking consistency of packages” on page 141).
- Check which programs are called by a package (see “Determining which programs are referenced” on page 140).
- Normalize packages to remove unnecessary required packages (see “Normalizing the list of required packages” on page 143).

From the VisualAge on Java Workbench window, make sure that the team leader is the current user. This ensures that the team leader will become the owner for all the packages and the owner for all the classes that are created when you commit to ENVY. See “Creating users and setting the current user” on page 116 for information on how to define and change users.

Action

From the VG Part Prerequisites View window, select the packages you want to commit. You can select individual packages or use the following to select packages:

- **Find Packages**
- **Select All**
- **Deselect All**

Be sure to select only packages that are to be placed in the same project.

You can specify:

- A package that has never been committed.
- A package that was previously committed to ENVY but which now indicates that it has been *Modified*. A *Modified* package is one that was committed to ENVY, but which you have added parts to or replaced parts within the package.

Note: You cannot commit a package that is already committed into ENVY unless it has been modified.

Select **Packages** and then **Selected** → **Commit Into ENVY**.

When you are prompted for a project name, enter the name of the project that is to contain all the selected packages. The package does not need to exist. If it does not, the MSL Migration Assistance Tool creates it for you.

If there were any missing parts listed in `notfound.pkg`, you can print a list of them by using the technique described in “Listing missing (not found) parts” on page 138.

Based on the MSL migration scenario that you are following, you should not delete the packages from the VG Part Prerequisites View window until you are certain you do not need them when you migrate additional MSLs. These committed packages can be used if you migrate additional MSLs to determine where duplicates exist and to help resolve the duplicates.

Description

If you are committing modified packages a warning message is displayed for each package. Select **OK** to commit the modified package to ENVY.

If there were any *Not Found* parts in any of the selected packages, a **Warning** message window is displayed. You can select **Cancel** to prevent committing any parts into ENVY if you want to resolve the *Not Found* parts first. You can select **OK** if you want to commit the existing parts to ENVY and resolve the *Not Found* parts later. Select **OK** if you know that the parts are in MSLs that you plan to migrate later (for example, if your scenario matches the one described in “Multiple subsystems with controlled duplicates” on page 89).

After the messages, a progress window is displayed. Committing the packages takes about 2 seconds per VAGen part.

When all packages have been committed, the VG Part Prerequisites View window is refreshed and each of the packages has the notation *Committed* beside it.

If there were any missing parts, a package called `notfound.pkg` was created and contains a list of the missing parts. The `notfound.pkg` was not committed to ENVY.

Action**Description**

See “Chapter 15. Completing the ENVY setup on Java” on page 151 and “Chapter 16. Completing your migration on Java” on page 159 for information on additional steps you might need to take.

Chapter 14. Using VAGen Import to migrate VAGen 2.x and Cross System Product non-GUI code to Java

To migrate VisualAge Generator 2.x or Cross System Product non-GUI code to VisualAge Generator 4.0 on the Java platform, use these steps:

1. Export your existing applications to external source format files. You must use your existing Cross System Product or VisualAge Generator product to create these .esf files.

Export one external source format file for each Java package you want to create. See the 2.x product documentation and the online help for information about how to create external source format files.

For VisualAge Generator 2.x or earlier, GUIs cannot be migrated to VisualAge Generator 4.0 on Java. Therefore, do not include GUIs in your external source format files. VisualAge Generator 4GL parts that were used in an existing GUI by being dropped on the freeform surface can be migrated to VisualAge Generator 4.0 along with other non-GUI parts.

If you are migrating from Cross System Product, also read the book *Migrating Cross System Product Applications to VisualAge Generator* (Version 3.1) for information about other tasks you must perform when migrating from Cross System Product to VisualAge Generator.

If you are changing from the OS/2 to the Windows NT development platform, be sure to review “Changing from OS/2 to Windows NT” on page 119 before you start to migrate.

Note: You should not modify the export files.

2. Start VisualAge Generator 4.0 on Java.
3. In VisualAge Generator 4.0, create a Java project and package to store the code you want to import:
 - a. From the VisualAge for Java Workbench window, open the context menu by clicking the right mouse button on an empty part of the workspace.
 - b. Select **Add->Project** from the context menu.
 - c. In the Add Project window, enter a name for the new project and select the **Finish** push button.
 - d. From the workspace, select the project you just created, and open the context menu.
 - e. From the context menu, select **Add->Package**.
 - f. In the Add Package window, enter a name for the new package and select the **Finish** push button.

Create one Java package for each .esf file that you created in step 1. For more information about creating packages, see the *VisualAge Generator User's Guide*.

4. In VisualAge Generator 4.0, from the VisualAge for Java Workbench, select **Workspace->Open VAGen Parts Browser** to open the VAGen Parts Browser.
5. From the VAGen Parts Browser window, select **Parts->Import/Export->VAGen Import**.
6. In the VAGen Import File Selection window, select one external source format (*.esf) file that you want to import and then select the **Open** button.
7. In the VAGen Import window, check to see if there are any parts listed in the Parts with errors list box. If there are, you need to debug the errors before those parts can be imported with the rest of the .esf file. When there are no parts with errors, proceed to the next step.
8. In the VAGen Import window, specify the name of your package in the Target package field. Move all parts to be imported from the Available parts list to the Selected parts list. Then select the **Import** push button. The selected parts are imported into the package you specified.
9. In the VAGen Import window, select the **New File** push button and then repeat steps 6, 7, and 8 for each external source format file you need to import. When you have finished importing your .esf files to VisualAge Generator 4.0, select the **Cancel** push button in the VAGen Import window to return to the VAGen Parts Browser window.
10. Version and release all classes, packages and projects for the existing applications you have imported.
11. Save your VisualAge Generator 4.0 workspace.
12. See "Chapter 15. Completing the ENVY setup on Java" on page 151 and "Chapter 16. Completing your migration on Java" on page 159 for information on additional steps you might need to take.

Chapter 15. Completing the ENVY setup on Java

The following sections describe specific tasks you might need to perform after running the MSL Migration Assistance Tool. These tasks include:

- “Versioning and releasing a VAGen part class”
- “Versioning and releasing a package” on page 152
- “Versioning a project” on page 153
- “Creating a Project List Part (PLP)” on page 154
- “Changing the owner of a project” on page 155
- “Assigning ownership of a VAGen part class” on page 155
- “Adding group members” on page 155
- “Changing the ownership of a VAGen part class” on page 156
- “Changing the owner of a package” on page 156

Note: ENVY provides a variety of ways of doing most tasks and supports multiple development scenarios. This chapter is intended to provide one way, but not necessarily the only way, of performing tasks related to migrating MSLs to ENVY.

Versioning and releasing a VAGen part class

After you have committed your production level VAGen parts to ENVY, you should version and release the VAGen part classes to provide a base line that matches the code that runs in your production system. The steps below describe how to version and release VAGen part classes for a package.

Action	Description
<p>From the VisualAge for Java Workbench window:</p> <ul style="list-style-type: none">• Select the Managing tab.• In the Projects pane, select a project.• In the Packages pane, select a package.• In the Types pane, select all the VAGen part classes. These are the classes that start with <i>VAGen</i> (for example, <i>VAGenRecords</i>).	<p>Note: In the Types pane, VAGen part classes that need to be versioned appear in the format:</p> <pre>>VAGenRecords(06/03/99 10:15:30 AM)</pre> <p>The date and time stamp next to the class name are also repeated in the status line at the bottom of the Workbench window, in the place where the version name is normally displayed.</p>

Action	Description
In the Types pane, press mouse button 2 and select Manage->Version .	The Versioning Selected Items window appears. Note: You must be the developer of a class to version the class.
<p>Select one of the following:</p> <ul style="list-style-type: none"> • Automatic (Recommended) to use the ENVY-determined defaults for each of the classes you are versioning. The default might be different for each class. • One Name to specify the same version name for all of the classes you are versioning. Use this option if you are migrating parts built with VisualAge Generator Templates or BW*Wizard or the business logic for those parts. • Name Each to specify a different version name for each of the classes you are versioning. <p>Also select Release selected items, and then select OK.</p>	<p>The Types pane in the VisualAge for Java Workbench window is refreshed and shows the version number for each VAGen part class.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. In the Types pane, VAGen part classes that have been versioned but not released appear in the format: >VAGenRecords 1.0 2. In the Types pane, VAGen part classes that have been versioned and released appear in the format: VAGenRecords 1.0 3. You must be the owner of a class to release the class.

Versioning and releasing a package

You should version and release your packages to provide a base line that reflects the level of code that you migrated to ENVY.

Action	Description
From the VisualAge for Java Workbench window, in the Packages pane of the Managing tab, select one or more packages to be versioned and released.	Note: In the Packages pane, packages that need to be versioned and released appear in the format: >trb.common.data.pkg (06/03/99 10:15:30 AM)
In the Packages pane, press mouse button 2 and select Manage->Version .	The Versioning Selected Items window appears. Note: You must be the owner of a package to version the package.

Action

Select one of the following:

- **Automatic (Recommended)** to use the ENVY-determined defaults for each of the packages you are versioning. The default might be different for each package.
- **One Name** to specify the same version name for all of the packages you are versioning. Use this option if you are migrating parts built with VisualAge Generator Templates or BW*Wizard or the business logic for those parts.
- **Name Each** to specify a different version name for each of the packages you are versioning.

Also select **Release selected items**, and then select **OK**.

Description

The **Packages** pane in the VisualAge for Java Workbench window is refreshed and shows the version number for each package.

Notes:

1. In the **Packages** pane, packages that have been versioned but not released appear in the format:
`>trb.common.data.pkg 1.0`
2. In the **Packages** pane, packages that have been versioned and released appear in the format:
`trb.common.data.pkg 1.0`
3. You must be the owner of the package or the owner of the project to release the package.

Versioning a project

To provide a base line for your projects that reflects the level of code you migrated to ENVY, you should version them.

Action

From the VisualAge for Java Workbench window, in the **Projects** pane of the **Managing** tab, select one or more projects to be versioned.

In the **Projects** pane, press mouse button 2 and select **Manage->Version**.

Select one of the following:

- **Automatic (Recommended)** to use the ENVY-determined defaults for each of the projects you are versioning. The default might be different for each project.
- **One Name** to specify the same version name for all of the projects you are versioning. Use this option if you are migrating parts built with VisualAge Generator Templates or BW*Wizard or the business logic for those parts.
- **Name Each** to specify a different version name for each of the projects you are versioning.

Then select **OK**.

Description

Note: In the **Projects** pane, projects that need to be versioned appear in the format:

```
TrbCommonDataProj (06/03/99 10:15:30 AM)
```

The Versioning Selected Items window appears.

The **Projects** pane in the VisualAge for Java Workbench window is refreshed and shows the version number for each project.

Notes:

1. In the **Projects** pane, projects that have been versioned appear in the format:
`TrbCommonDataProj 1.0`
2. You must be the owner of the project to version the project.

Creating a Project List Part (PLP)

A Project List Part (PLP) is a VisualAge Generator generation options part. It is used to identify projects to load before the containing project is loaded. The PLP is the VisualAge Generator on Java way of implementing VisualAge Generator on Smalltalk required maps.

A PLP allows you to define a group of project editions or versions that should all be loaded together into your workspace. A PLP is used for storing a list of /PROJECT generation options. In the PLP, you define one /PROJECT option for each project in the group you want to load together. For example, you might have a project that defines the package versions that are currently in your production system.

Use these steps to create a Project List Part that you can use to load a group of projects before loading this project:

1. Create an open edition of the project by selecting the project and then selecting **Manage->Create Open Edition** from the context menu.
2. Select the open edition of the project and then select **Add->Package** from the context menu.
3. When the Add Package Wizard dialog is displayed, select the **Create a default package** radio button to create a default package for the project.
4. Select the default package in the Workbench window and select **Packages->Manage->Release** to release the default package into the project.
5. Open the VAGen Parts Browser, and create a new generation options part in this package with the following name:

VAGEN_project_name.PLP

where:

VAGEN_

Is the required prefix for all PLP parts

project_name

Is the name of the project, with all blanks converted to underscores (_).

If this name is longer than 22 bytes, the name must be truncated on the right to 22 bytes. If the name is mixed SBCS and DBCS, and the 22nd byte is the first byte of a DBCS character, the name should be truncated to 21 bytes.

.PLP Is the required suffix for all PLP parts.

6. Insert the desired /PROJECT options in the PLP options part.
7. Version and release the class *VAGenOptions* in the default package.

8. Version the default package and project.

Note: For more information about the /PROJECT generation option, see “Generation options” on page 56 and the *VisualAge Generator Generation Guide*.

Changing the owner of a project

Each project can have a different owner. To change the owner assigned to a project during migration, follow these steps:

Action	Description
From the VisualAge on Java Workbench window: <ul style="list-style-type: none">• Select the Managing tab.• In the Projects pane, select the project for which you want to change the owner.	The Change Owner window is displayed showing the available users. The current owner is highlighted.
Select the Projects menu, select Manage , and then select Change Owner .	
Select the new owner and select OK .	The new project owner is displayed in the Project Owner pane. Note: You must be the owner of a project to change the owner of that project.

Assigning ownership of a VAGen part class

Each VAGen part class within a package can have a different owner. Although developers can version a VAGen part class, the class owner controls the release of VAGen part classes into the package. The following sections describe how to add members to the group for a package and how to change the owner for a class within a package.

Adding group members

To change the owner of the class, the new owner must be a member of the group. Group members must be defined for one package at a time.

Action	Description
From the VisualAge on Java Workbench window, select the package for which group members are to be added.	The current group members are shown in the Package Group Members pane. The current group owner is indicated by a > beside the name.
Select the Packages menu and then select Manage->Add User to Group .	The Add Users window showing all defined users for the package is displayed.

Action	Description
Select a user to add to the group, and then select OK . Use the Ctrl key to select several users before selecting OK .	All the new users are added to the list in the Package Group Members pane. Note: You must be the owner of a package to add group members for that package.

Changing the ownership of a VAGen part class

The same owner can be assigned for multiple classes within a single package at the same time. During migration, it is easiest to assign the same owner to all the VAGen part classes within a package. The owner of these classes might also (most likely) be the owner of the containing package.

Action	Description
From the VisualAge on Java Workbench window: <ul style="list-style-type: none"> • Select the Managing tab. • In the Packages pane, select the package for which VAGen part class owners are to be assigned. • In the Types pane, select the VAGen part classes for which the same owner is to be assigned. 	A Change Owner window is displayed showing the current group members for the package. The current owner is highlighted. Note: You must be a package group member to change the owner of a class within the package.
Select the Types menu and then select Manage->Change Owner .	
Select the new owner and select OK .	The new owner is shown in the Type Owner pane.

Changing the owner of a package

Each package can have a different owner. The owner of a package can create an edition of the package and can version the package.

To change the owner of the package, the new owner must be a member of the group. See “Adding group members” on page 155 for information on adding users to a group.

Action	Description
From the VisualAge on Java Workbench window: <ul style="list-style-type: none"> • Select the Managing tab. • In the Packages pane, select the package for which you want to change the owner. 	A Change Owner window is displayed showing the current group members for the package. The current owner is highlighted. Note: You must be the owner of a package to change the owner of that package.
Select the Packages menu and then select Manage->Change Owner .	

Action

Select the new owner and select **OK**.

Description

The new owner is indicated by a > beside the name in the **Package Group Members** pane.

Chapter 16. Completing your migration on Java

The following sections describe specific tasks you might need to perform to complete your migration to VisualAge Generator 4.0. These tasks include:

- “Defining control information”
- “Generating Programs” on page 160
- “Importing work-in-progress” on page 161
- “Migrating VSAM files” on page 163

In addition, “Chapter 33. Hints and tips on Smalltalk” on page 345 provides information to help you use the VisualAge Organizer and VAGen Parts Browser windows.

Note: ENVY provides a variety of ways of doing most tasks and supports multiple development scenarios. This chapter is intended to provide one way, but not necessarily the only way, of performing tasks related to migrating MSLs to ENVY.

Defining control information

Control information that is needed for test and generation must be stored in ENVY packages. This control information consists of:

- Generation options
- Linkage table
- Resource associations
- Bind control information
- Link edit information

The technique for migrating your existing control information is the same for each of the control files. The only difference is the part type that you specify when you create the part.

Action	Description
Make sure you have open editions of the project and package to which you want to add a control information part.	The project and package show the date and time stamp when the edition of each was created.
If you do not have open editions, from the VisualAge on Java Workbench window: <ul style="list-style-type: none">• Select the project, select the Projects menu, and then select Manage->Create Open Edition.• Select the package, select the Packages menu, and then select Manage->Create Open Edition.	

Action	Description
From the VAGen Parts Browser, select the VAGen Parts menu and then Add->New Part .	The New VAGen Part window is displayed.
Specify the name of the new part.	An editor window is displayed.
Select the Other radio button and then from the drop-down list, select the type that corresponds to the control information you need to add:	
<ul style="list-style-type: none"> • <i>Generation Options</i> • <i>Linkage Table</i> • <i>Resource Associations</i> • <i>Bind Control</i> • <i>Link Edit</i> 	
From the drop-down combination box, select the package to which you want to add the part. The drop-down list shows packages with open editions and packages that already contain the type of control information you specified.	
Select the OK push button.	
To load the control information from an existing file, select File and then Read From File .	A File Specification window is displayed.
Select the drive, directory, and file name from which you want to load an existing control file. Double-click on the file name or select the Open push button to load the file.	<p>The text from the file you specified appears in the editor window.</p> <p>Note: Be sure to review your options for any that need to be changed. For example, /OPTIONS, /LINKAGE, /RESOURCE, /BIND, and /LINKEDIT specified a directory in VisualAge Generator 2.2. Now they should specify a part name that is stored in ENVY.</p>
If you do not have an existing file, you can use the editor to specify your control information. If you loaded an existing file, you can also use the editor to make changes to the control information.	
Close the editor window and specify Yes when asked if you want to save the changes.	The VAGen Parts Browser is displayed.
Press F5 to refresh the contents of the VAGen Parts Browser window.	

Generating Programs

After you complete the migration of a group of MSLs (for example, after migrating your production MSLs), you should generate all the programs for your target environments. Then test the results in the runtime environments. This helps to ensure that you have migrated the correct version of your code.

Refer to the *VisualAge Generator Generation Guide* for more information on how to generate your programs.

Note:

1. Any programs for the C++ target environments **must** be regenerated for VisualAge Generator 4.0. There is no coexistence of C++ runtime services between VisualAge Generator 2.2 and 4.0. In addition, it is **strongly recommended** that you regenerate all programs for the COBOL environments to be sure that you migrated the correct level of code.
2. For CICS OS/2, the default *parmform* option in the linkage table was *COMMDATA*. With VisualAge Generator 4.0, the new option *COMMPTR* is the default. Therefore, if you never specified linkage tables for CICS OS/2, you might need a linkage table now.

Importing work-in-progress

To migrate your work-in-progress MSLs, use **VAGen Import**.

Action	Description
Make sure you have an open edition of the project and package for which you want to change or add parts.	The package shows the date and time stamp when the edition was created.
If you do not have open editions, the project owner and package owner can open editions from the VisualAge on Java Workbench window by:	
<ul style="list-style-type: none">• Selecting the project, selecting the Projects menu and then selecting Add->Project.• Selecting the package, selecting the Packages menu and then selecting Add->Package.	
Note: During migration, you probably want to open an edition of the project and package because you will be versioning after importing the external source format file from each work-in-progress MSL. After migration, you can use a scratch edition when you import if you will not be adding any classes (no new VAGen part classes).	

Make sure you are a group member of all the packages for which you want to change or add parts.

If you are not a group member, the package owner can add you to the group by following the steps in “Adding group members” on page 155.

Action	Description
If some of the parts you will be importing should be placed into new packages, create the new packages.	The New Package window is displayed.
From the VisualAge on Java Workbench window, select the Packages menu and Add->Package .	
Type the name of the package you want to create.	The VisualAge on Java Workbench window is displayed with the new package listed in the Packages pane.
Select the OK push button.	
From the VisualAge on Java Workbench window, select the Workspace menu and then Open VAGen Parts Browser .	The VAGen Parts Browser window is displayed.
From the VAGen Parts Browser window, select Parts and then Import .	The VAGen Import File Selection window is displayed, requesting information about the external source format file that you want to import.
Specify the drive, directory, and file name and then press the OK push button.	The VAGen Import window is displayed. The Parts with errors pane lists any parts that encountered problems during external source format validation. The Available parts pane lists the parts that successfully passed the external source format validation.
Specify the Destination for duplicate parts and the Target package .	<p>Destination for duplicate parts specifies how duplicate parts (parts that are already in your workspace) are to be treated. When you import your work-in-progress, you should select the Defined package radio button as the Destination for duplicate parts. This means that any part listed in the Selected Parts field that already exists in your workspace is replaced in the package in which it already exists. This preserves the package organization that you created in the sandbox with the MSL Migration Assistance Tool.</p> <p>Target package is the default package into which you want to place the parts. When you select Defined package as the method for handling duplicates, only new parts are imported into the target package.</p>

Action	Description
<p>Select parts from the Available parts pane and move them to the Selected parts pane using the >> push button.</p> <p>When you select the Import push button, only parts in the Selected parts pane are imported. This allows you to import some parts into one target package and another group of parts from the same external source format file into a different target package.</p>	<p>After you select the Import push button, the selected parts are imported and the VAGen Import window is displayed again. The parts you imported are removed from the Selected parts pane and also do not appear in the Available parts pane. This enables you to see what parts from the external source format file have not yet been processed.</p>

Migrating VSAM files

If you are migrating from Cross System Product, you might have VSAM files that you need to access during test. Copies of these files must be moved to the workstation for use with the Interactive Test Facility. To migrate these VSAM files, do the following:

- If you use variable-length VSAM files on an MVS host system, see the *VisualAge Generator Installation Guide* for information on how to upload and install a VisualAge Generator utility that is required to prepare these files for download.
- For VM, VSE, and MVS host systems, see *Migrating Cross System Product Applications to VisualAge Generator* (SH23-0244-01) for information on how to REPRO and download your VSAM files.
- For information on how to run the VSAM conversion utility for your downloaded MVS files, from the VAGen Parts Browser window, select **Tools** and then **Data File Conversion**. Then select the **Help** push button.

Converting an RTABLE to a Linkage Table

If you have been using the VisualAge Generator middleware RTABLE for communications routing, the RTABLE entries must be moved from the RTABLE to a linkage table. The following example shows a mapping of RTABLE entries to linkage table entries:

```

RTABLE
app1 - - - - - 1u2 LU2C - 1
app2 - - - - - 1u2 LU2C - 1
app3 - - - - - 1u2 LU2C - 1
app4 - - - - - 1u2 LU2B - 1
$ANY - - - - - 1u2 LU2K - 1

LINKAGE TABLE
:calllink applname=app1 linktype=remote remotecomtype=LU2
serverid=LU2C.
:calllink applname=app2 linktype=remote remotecomtype=LU2

```

```
serverid=LU2C.  
:calllink applname=app3 linktype=remote remotecomtype=LU2  
serverid=LU2C.  
:calllink applname=app4 linktype=remote remotecomtype=LU2  
serverid=LU2B.  
:calllink applname=* linktype=remote remotecomtype=LU2  
serverid=LU2K.
```

Part 3. Migrating from VAGen 3.x to VAGen 4.0 on Smalltalk

Chapter 17. General migration considerations for VAGen 3.x to 4.0 on Smalltalk 167

Migration paths	167
Overview of the V3 to V4 Migration Tool on Smalltalk.	168
Automatic conversions during 4.0 migration	169
Migrating GUIs	170
Migrating applications, subapplications, and configuration maps	170
Migrating VAGen Templates	171
Establishing naming conventions.	172
Assigning ownership	173
Using subapplications	173
Storing control information	173
Dual maintenance.	173
Migrating from OS/2 to Windows NT	174

Chapter 18. Pre-migration checklist 175

Chapter 19. Using the V3 to V4 Migration Tool to migrate VAGen 3.x code and VAGen Templates parts on Smalltalk . . . 177

Setting V3 to V4 Migration Tool options on Smalltalk.	178
Selection Criteria options	179
Library Management options	181
Ownership options	182
Selecting and migrating applications and configuration maps	183
Working with the status log	185
Resetting the V3 to V4 Migration window	187
Resetting Migration Status Information	188

Chapter 20. Using VAGen Import to migrate VAGen 3.x non-GUI code to Smalltalk 189

Chapter 21. Completing the ENVY setup on Smalltalk 191

Versioning and releasing a view or a VAGen part class	191
Versioning an application	192

Creating a configuration map	193
Adding a required map to a configuration map	194
Versioning a configuration map	195
Changing the manager of a configuration map	195
Testing a configuration map	196
Assigning ownership of a VAGen part class	196
Adding group members	196
Changing the ownership of a VAGen part class	197
Changing the manager of an application	197

Chapter 22. Completing your migration on Smalltalk 199

Defining control information	199
Generating programs	199
Importing work-in-progress	199
Recreating ITF resource association information	200
Converting an RTABLE to a Linkage Table	200

Chapter 17. General migration considerations for VAGen 3.x to 4.0 on Smalltalk

Consider the following if you are migrating from VisualAge Generator 3.x to VisualAge Generator 4.0 on Smalltalk:

- “Migration paths”
- “Overview of the V3 to V4 Migration Tool on Smalltalk” on page 168
- “Automatic conversions during 4.0 migration” on page 169
- “Migrating GUIs” on page 170
- “Migrating applications, subapplications, and configuration maps” on page 170
- “Migrating VAGen Templates” on page 171
- “Establishing naming conventions” on page 172
- “Assigning ownership” on page 173
- “Using subapplications” on page 173
- “Storing control information” on page 173
- “Dual maintenance” on page 173
- “Migrating from OS/2 to Windows NT” on page 174

Also see “Chapter 18. Pre-migration checklist” on page 175 before you begin to migrate code to VisualAge Generator 4.0.

Migration paths

Depending on your current platform and whether you want to migrate to Smalltalk, Java, or both, different migration paths are available and different considerations apply.

Table 10 on page 168 gives a brief overview of the migration options available for the Smalltalk platform. See the referenced chapters for step-by-step procedures for each migration option.

Table 10. Migration Options

Migrating from	Tools to Use	Details on Using the Tools
VisualAge Generator 3.x (GUI and non-GUI)	<ul style="list-style-type: none"> • V3 to V4 Migration Tool • VAGen Import 	<ul style="list-style-type: none"> • “Chapter 19. Using the V3 to V4 Migration Tool to migrate VAGen 3.x code and VAGen Templates parts on Smalltalk” on page 177 • “Chapter 20. Using VAGen Import to migrate VAGen 3.x non-GUI code to Smalltalk” on page 189
VisualAge Generator Templates 3.x	V3 to V4 Migration Tool	“Chapter 19. Using the V3 to V4 Migration Tool to migrate VAGen 3.x code and VAGen Templates parts on Smalltalk” on page 177
VisualAge Generator 4.0 on Java parts	VAGen Import	“Chapter 34. Sharing VAGen 4.0 parts between Java and Smalltalk” on page 351

Overview of the V3 to V4 Migration Tool on Smalltalk

Migration from VisualAge Generator 3.x to 4.0 on Smalltalk is a two step process:

- Import configuration maps and/or applications from the VisualAge Generator 3.x library into the 4.0 library using the normal Smalltalk import process.
- Run the V3 to V4 Migration Tool to convert the 4GL parts and GUIs that use 4GL parts for use in VisualAge Generator 4.0

The V3 to V4 Migration Tool on Smalltalk does the following:

- Migrates versioned configuration maps or applications from VisualAge Generator 3.x
- Sets flags in the 4.x library to indicate what has been migrated
- Provides options to control configuration map or application selections, project and package naming conventions, and version, release, and ownership information.

Automatic conversions during 4.0 migration

VisualAge Generator 4.0 automatically makes the following changes to your applications during migration:

- Process and statement group parts are converted to function parts.
- References to processes and statement groups are converted to the new syntax requirements for functions with no parameters.
- PERFORM statements and Unconditional Branch statements are no longer supported and are migrated to Function Invocation statements.
- Subscript parentheses are changed to brackets in VisualAge Generator item names in the following places:
 - 4GL statements in functions (processes and statement groups)
 - Host variable names in SQL statements
 - Comparison value item in DL/I specifications
 - EZEDLPCB is used in a called parameter list
- Calls to EZE service routines are converted to the corresponding function invocation statement. A statement to set the value of EZEREPLY is also added before the function invocation.
- VisualAge Generator-supplied string and math functions are converted from the CALL statement to a function invocation statement or to an assignment statement that contains the function as the source of the assignment. A statement to set the value of EZEREPLY is also added before the function invocation.

VisualAge Generator Templates 4.0 automatically makes the following changes to your applications during migration:

- **Definition** and **Generation Parameters** of a VAGT instance are stored in two different VAGTemplates part classes. (In version 3.x, these two types of descriptions are stored in the same VAGTemplates part class.)
- **Definition Extensions** for all VAGT entities are stored in an appropriate VAGTemplates part class. (In version 3.x, they are stored in the same VAGTemplates part class as all the instances of the same VAGT entity type.)

In addition, the names of part classes, control information files, and VisualAge Generator palette parts are changed during migration to VisualAge Generator 4.0:

- Table 20 on page 361 shows the changes made to VAGen part class names during 4.0 migration.
- Table 21 on page 361 shows the changes made to VAGen control information part class names during 4.0 migration.
- Table 22 on page 362 shows the changes made to VisualAge Generator palette parts names during 4.0 migration.

- Table 23 on page 362 shows the changes made to VAGen Templates part class names and repartition during 4.0 migration.

Migrating GUIs

VisualAge Generator 4.0 provides two ways to migrate existing GUIs to the Smalltalk platform.

1. Use the V3 to V4 Migration Tool. See “Chapter 19. Using the V3 to V4 Migration Tool to migrate VAGen 3.x code and VAGen Templates parts on Smalltalk” on page 177 for step-by-step instructions on how to migrate your GUIs using the V3 to V4 Migration Tool.
2. Use ENVY import:
 - a. From VisualAge Generator 4.0, use these steps to import the applications and configuration maps from your 3.x library:
 - 1) From the VisualAge Organizer window, select **Applications->Import/Export->Import Applications** to import all the application versions that you want to migrate.
 - 2) From the System Transcript window, select **Tools->Browse Configuration Maps** to open the Configuration Maps Browser. From the Configuration Maps Browser window, select **Names->Import** to import all the configuration map versions that you want to migrate.
 - b. On the VisualAge Organizer window, select the applications you want to migrate, and then select **Migrate VAGen GUIs**.

Note: You must run the V3 to V4 Migration Tool to convert any applications that contain GUIs that use 4GL parts.

Migrating applications, subapplications, and configuration maps

Your current structure of applications, subapplications, and configuration maps is preserved during migration to VisualAge Generator 4.0 on Smalltalk.

Note: The V3 to V4 Migration Tool loads the configuration maps and applications into your image as part of the migration process. Therefore, if you have used multiple configuration expressions for configuration maps and/or applications, you must migrate on each of the platforms that corresponds to one of your configuration expressions. Required maps, prerequisite applications, and subapplications can only be migrated when their corresponding configuration expression evaluates to true.

Migrating VAGen Templates

The following table outlines considerations that apply when you select VAGen Templates for migration using the V3 to V4 Migration Tool:

Table 11. VAGen Template Migration to VisualAge Generator 4.0 on Smalltalk

VisualAge Generator 3.x	VisualAge Generator 4.0 on Smalltalk
Standard Generators	The standard generators from VAGen Templates 3.x are not migrated. They have been rewritten for VisualAge Generator 4.0 on Smalltalk to make use of the new functions.
Customized Generators	Your customized generators from VAGen Templates 3.x are not migrated. You must reapply your customization to the new standard generators for VisualAge Generator 4.0 on Smalltalk. To ease migration for users who customized the VisualAge Generator 3.x generators, the standard generators from VisualAge Generator 3.1 are shipped with VisualAge Generator Templates Customizer 4.0 (Smalltalk Version). These 3.x obsolete standard generators will produce VisualAge Generator 4.0 components (for example, functions instead of statement groups or processes). You can reapply your customization to the 3.x obsolete standard generators or to the new, enhanced standard generators.
Specifications	The VAGen Templates specifications are migrated automatically by the V3 to V4 Migration Tool. If you want to make use of any of the new functions in the new generators, you can migrate your specifications and then regenerate using the new generators. For example, you could generate using a migrated specification and one of the new Web generators.

Table 11. VAGen Template Migration to VisualAge Generator 4.0 on Smalltalk (continued)

VisualAge Generator 3.x	VisualAge Generator 4.0 on Smalltalk
VAGen Templates-generated 4GL code	The 4GL code generated by VAGen Templates 3.x migrates like any other 4GL code. The VAGen Templates generated components include “traceability” information that is used only by VAGen Templates. This information is migrated automatically by the V3 to V4 Migration Tool. After migrating, the 4GL code is ready to be used. Your custom business logic is already incorporated and reacts as a brand new one. This means that you just need to generate the 4GL code using the ‘normal’ generation option. (The ‘override’ option would erase your custom business logic.)
VAGen Templates-generated GUI code	The GUIs generated by VAGen Templates 3.x migrate like any other GUI code. Therefore, all your custom business logic is preserved in the migrated GUI parts. But, in order to update your GUI parts using the new V4 templates, generate your GUI parts with the ‘normal’ generation option.

Establishing naming conventions

You probably already have naming conventions for parts like processes, records, data items, and so on that you are migrating from VisualAge Generator 3.x. You can continue to use your existing naming conventions, because VisualAge Generator 4.0 retains existing part names during migrations from VisualAge Generator 3.x.

However, VisualAge Generator does change some existing part *types*, as well as making some syntax conversions. For example, process parts and statement group parts are both changed to function parts. Therefore, you might want to establish a naming convention for function parts that is similar to your conventions for processes and statement groups.

See “Automatic conversions during 4.0 migration” on page 169 for a list of changes that VisualAge Generator makes to your applications during migration.

When you use the V3 to V4 Migration Tool to migrate applications with part types that are changed during migration, you can decide whether the migrated applications are automatically versioned and released during migration. See “Setting V3 to V4 Migration Tool options on Smalltalk” on page 178 for more information.

Assigning ownership

Using the V3 to V4 Migration Tool, you can decide whether you want to retain the existing ownership structure for applications, subapplications, and configuration maps. When you set the V3 to V4 Migration Tool options, you can choose to maintain the existing ownership structure or change the ownership to the current user. See “Ownership options” on page 182 for more information.

Using subapplications

Your existing subapplication structure is preserved during migration. However, you still need to consider the ownership issues discussed in “Assigning ownership”.

Storing control information

Control information consists of generation options, linkage table, resource association, bind control, and link edit command parts. In VisualAge Generator 3.x, the control information is stored in ENVY parts. When you migrate to 4.0 on Smalltalk, the control information is preserved. No special migration considerations apply.

Dual maintenance

The external source format file for 4GL parts that you export from VisualAge Generator 4.0 is not compatible with the external source format file for VisualAge Generator 3.x. Therefore, if you migrate a subsystem that shares common parts with a subsystem that you will migrate at a later time, you have the following alternatives for maintenance of the common parts:

1. Maintain the common parts in VisualAge Generator 3.x, and when you are satisfied with the changes:
 - a. Export an external source format file from the 3.x application for the changed parts.
 - b. Import the external source format file into VisualAge Generator 4.0 on Smalltalk using the **Defined application** radio button so the changes will go into the same ENVY application in which the parts are already located.

2. Make the same changes to both the parts in VisualAge Generator 4.0 on Smalltalk using VisualAge Generator 4.0 and to the corresponding 3.x parts using VisualAge Generator 3.x.

With VisualAge Generator 4.0, you cannot export external source format files for views. Therefore, the only option for views is to make the same changes to the view using VisualAge Generator 4.0 and to the corresponding view in VisualAge Generator 3.x. If the view does not contain any 4GL parts, you could use the Smalltalk export and import facilities to transfer the modified view from VisualAge Generator 3.x to VisualAge Generator 4.0.

Migrating from OS/2 to Windows NT

Note: If you use a DBCS code page, skip this section. Code page conversion is not required for DBCS code pages.

If you are changing from the OS/2 to the Windows NT development platform, do the following:

- Using VisualAge Generator 3.x on the OS/2 development platform, create a test application containing one process or statement group. In this process or statement group, as comment lines, include all the special characters that you use. Version the test application.
- Migrate the test application using the V3 to V4 Migration Tool to the Windows NT development platform.
- If all the special characters were transferred correctly, you can use the V3 to V4 Migration Tool to do your migration.
- If some of the special characters were not transferred correctly, you must use VAGen Import to migrate your code. See the following sections:
 - “Chapter 20. Using VAGen Import to migrate VAGen 3.x non-GUI code to Smalltalk” on page 189
 - “Changing from OS/2 to Windows NT” on page 298

If you plan to use VAGen Import to migrate your code, you can run a similar test using VAGen Import instead of the V3 to V4 Migration Tool to determine whether you can skip the code page conversion step described in “Changing from OS/2 to Windows NT” on page 298.

Chapter 18. Pre-migration checklist

1. Before you migrate, you should first read the following:
 - “Chapter 17. General migration considerations for VAGen 3.x to 4.0 on Smalltalk” on page 167
 - “Chapter 19. Using the V3 to V4 Migration Tool to migrate VAGen 3.x code and VAGen Templates parts on Smalltalk” on page 177
 - VisualAge Generator 4.0 readme file
2. Load any features you need for migration.
3. Save a clean copy of your VisualAge Smalltalk image. This clean copy should not contain any of your application code. Copy *abt.icx* to *abtclean.icx*, and store the clean copy on your LAN so that all developers have access to it. Saving copies of the *hpt.ini*, *abt.ini*, and *mgr50.dat* files is also recommended.
4. Check with IBM support to see if there are any fixes available for the VisualAge Generator 4.0 V3 to V4 Migration Tool.
5. Contact your local IBM representative to learn more about VisualAge Generator service offerings that can help you with migration.
6. Be sure that the configuration maps and applications that you plan to migrate have been versioned on VisualAge Generator 3.x.

Chapter 19. Using the V3 to V4 Migration Tool to migrate VAGen 3.x code and VAGen Templates parts on Smalltalk

VisualAge Generator 4.0 on Smalltalk includes a V3 to V4 Migration Tool for migrating 3.x code and VAGen Templates parts to 4.0. The V3 to V4 Migration Tool allows you to migrate your 3.x applications and configuration maps to VisualAge Generator 4.0 on Smalltalk, using options that apply across migration operations. You can change the options as needed. For example, you can set options that apply for one group of applications and configuration maps, migrate that group, and then set different options for the next group to be migrated.

Use these steps to start the V3 to V4 Migration Tool:

1. Start VisualAge Generator 4.0 on Smalltalk.
2. From the VisualAge Organizer window, select **VAGen Parts->Parts Browser**. The VAGen Parts Browser window is displayed.
3. From the VAGen Parts Browser window, select **Tools->Migration->V3 to V4 Migration**. The main V3 to V4 Migration Tool window, called V3 to V4 Migration, is displayed.

Figure 21 shows the V3 to V4 Migration window.

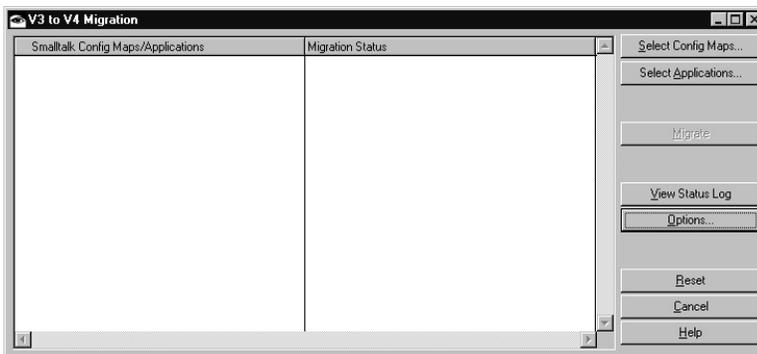


Figure 21. V3 to V4 Migration window

From the V3 to V4 Migration window, you can perform the following tasks:

- Set migration options
- Select and migrate configuration maps and applications
- View and work with the status log

- Reset the V3 to V4 Migration window

These tasks are described in the following sections.

Setting V3 to V4 Migration Tool options on Smalltalk

The V3 to V4 Migration Tool allows you to set migration options that can be applied for all applications and configuration maps that you later select for migration.

When you select applications and configuration maps to be migrated to VisualAge Generator 4.0, the V3 to V4 Migration Tool uses some of the options you set in the Migration Options window to display the applications and configuration maps that are available for migration. Other options are used during the migration for versioning and ownership assignment. You can migrate one group of applications or configuration maps using one set of options, and then change the options before migrating another group.

To access the Migration Options window from the V3 to V4 Migration window, select the **Options** push button.

Figure 22 on page 179 shows the Migration Options window.

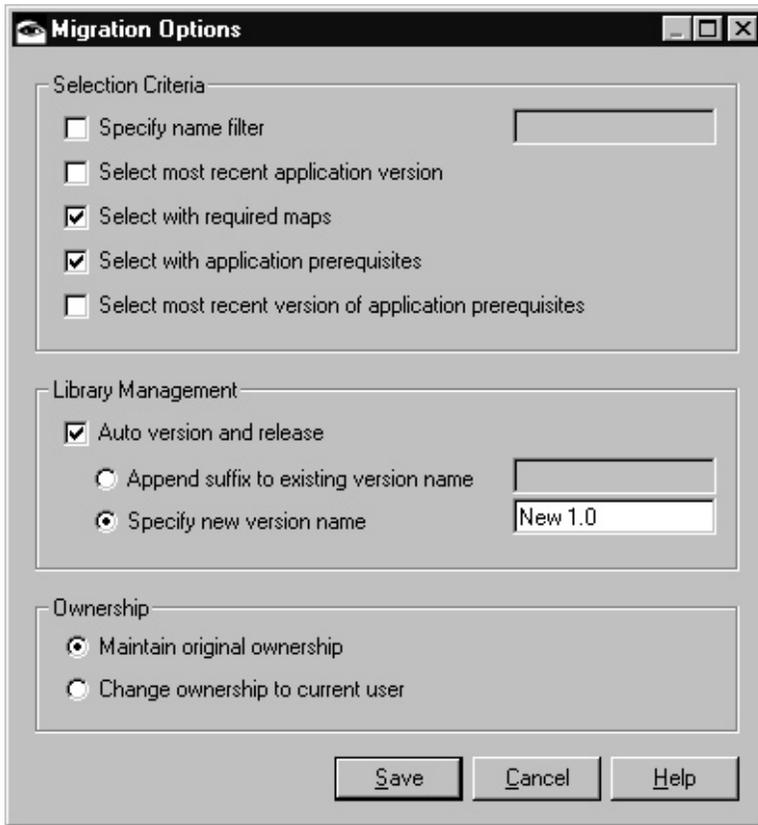


Figure 22. Migration Options window

In the Migration Options window, you can set defaults that are applied to all applications and configuration maps you later select for migration. The Migration Options window is divided into groups of options. These option groups are described in the following sections.

Selection Criteria options

The Selection Criteria options allow you to specify options that aid in application and configuration map selection when you later select **Select Applications** or **Select Config Maps** from the V3 to V4 Migration window.

Table 12. Selection criteria migration options

Option	Migration processing
Specify name filter	Allows you to use a naming convention to select a group of applications or configuration maps for migration. When this box is checked, the Filter prompter window is displayed each time you later select the Select Applications and Select Config Maps push buttons on the V3 to V4 Migration window.
Select most recent application version	With this box checked, when you later select an application for migration, the most recent version is automatically selected for you.
Select with required maps	With this box checked, when you later select a configuration map for migration, all the required configuration maps for your selected configuration map are automatically loaded into the V3 to V4 Migration Tool and your 4.0 image along with your selected configuration map. Note: If a configuration map has more than one configuration expression, VisualAge Generator 4.0 on Smalltalk uses the expression that evaluates to true when loading the configuration map. In this case, only the required maps that correspond to the true configuration expression are migrated.

Table 12. Selection criteria migration options (continued)

Option	Migration processing
Select with application prerequisites	<p>With this box checked, when you later select an application for migration, all the user-created prerequisite applications that could contain VAGen parts are automatically loaded into the V3 to V4 Migration Tool and your 4.0 image along with your selected application. If more than one version of a prerequisite application is available, when you later select an application for migration, you will be prompted to select the version you want.</p> <p>Note: If an application has more than one configuration expression, VisualAge Generator 4.0 on Smalltalk uses the expression that evaluates to true when loading the application. In this case, only the prerequisite applications that correspond to the true configuration expression are migrated.</p>
Select most recent version of application prerequisites	<p>With this box checked, when you later select an application for migration, the most recent version of each prerequisite for that application is automatically loaded into the V3 to V4 Migration Tool along with your selected application.</p>

Library Management options

The Library Management options allow you to specify versioning and releasing conventions for migrated configuration maps, applications, and classes.

Table 13. Library management migration options

Option	Migration processing
Auto version and release	<p>All configuration maps, applications, and classes are automatically versioned and released during migration.</p> <p>When you check this box, you can also select Append suffix to existing version name or Specify new version name.</p> <p>If you do not check this option, all editions of migrated configuration maps, applications, and classes are left open, and developers and owners must version and release their own configuration maps, applications, and classes after migration. Also, if you do not check this box, you cannot select either of the sub-options under this option.</p>
Append suffix to existing version name	<p>Adds a suffix to the existing version name. In the text box to the right of this option, you can specify the suffix to be appended. Using a suffix can help in associating the migrated version with the original version and in determining which other versions you still need to migrate. If you check this box but do not specify a suffix to append, the error message <code>Text required if selected</code> is displayed.</p>
Specify new version name	<p>Enables you to specify a new version name to be assigned for all your migrated configuration maps, applications, and classes. If you check this box but do not specify a version name, the error message <code>Text required if selected</code> is displayed.</p>

Ownership options

The Ownership options allow you to specify whether the ownership assignments for your current configuration maps, applications, and classes will be preserved during migration or changed.

Note: Regardless of the ownership option you select, you can select the **Auto version and release** option under Library Management.

Table 14. Ownership migration options

Option	Migration processing
Maintain original ownership	Allows you to maintain the existing ownership structure for all migrated configuration maps, applications, and classes.
Change ownership to current user	<p>Assigns ownership for all migrated configuration maps, applications, and classes to the user who performs the migrations.</p> <p>After the migration process completes, you can change the ownership back to the original owner or any other owner.</p>

Selecting and migrating applications and configuration maps

Follow these steps to migrate applications and configuration maps using the V3 to V4 Migration Tool:

Note: You cannot use the V3 to V4 Migration Tool to migrate an open edition of a 3.x application or configuration map. You must use VisualAge Generator 3.x to version and release each application and configuration map that you want to migrate to VisualAge Generator 4.0.

1. From VisualAge Generator 4.0, use these steps to import the applications and configuration maps from your 3.x library:
 - a. From the VisualAge Organizer window, select **Applications->Import/Export->Import Applications** to import all the application versions that you want to migrate.
 - b. From the System Transcript window, select **Tools->Browse Configuration Maps** to open the Configuration Maps Browser. From the Configuration Maps Browser window, select **Names->Import** to import all the configuration map versions that you want to migrate.
2. Open the VAGen Parts Browser and then select **Tools->Migration->V3 to V4 Migration** to start the V3 to V4 Migration Tool.
3. Select the **Options** push button to set migration preferences that will be used during the migration process. (Some options can be changed again before you select the **Migrate** push button.)
4. Display a list of applications or configuration maps available for migration by selecting one of the following:
 - Select the **Select Applications** push button if you want to migrate applications.

- Select the **Select Config Maps** push button if you want to migrate configuration maps.

Note: You cannot select a mixture of configuration maps and individual applications for migration at the same time. You can migrate a group of applications or a group of configuration maps, but not both together. When you select one of these buttons, the other button is disabled until you select the **Reset** button. When you select **Reset**, the list of selections currently in the V3 to V4 Migration window is deleted, and both **Select Applications** and **Select Config Maps** are available for selection again.

5. If you selected the **Specify name filter** option on the Migration Options window, the Filter prompter window is displayed. Enter a case-sensitive prefix to be used in filtering the application list. For example, if you want to see only applications that begin with Xyz, such as XyzPayApp and XyzEmployeeApp, enter **Xyz** in this window. You can use the * (asterisk) wildcard in specifying the search string. For example, use **Xyz*Sample*** to find applications such as Xyz03SampleApp, XyzMMediaSampleApp, and XyzPaySampleOpnsApp.
6. On the Selection Required window, select an application or configuration map and a version to be migrated, and then select the >> arrow button to add your selection to the Selected Versions column.

After you have selected a version for each application or configuration map that you want to migrate, select **OK**.

For applications only, if you selected the **Select most recent application version** option under the Selection Criteria section of the Migration Options window, this window shows only a single-column list of applications, because the versions have been selected for you. You can select multiple applications for migration. When you are finished selecting applications, select **OK**.

Note: The V3 to V4 Migration Tool loads the selected configuration maps and applications into your image as part of the migration. Therefore, only select configuration maps or applications that can be loaded together.

7. A populated version of the V3 to V4 Migration window is displayed. It lists the applications and configuration maps you selected for migration. Before migrating, you can add more applications or configuration maps to this list, using the above steps.

You can delete any application or configuration map from the list to be migrated by selecting it in the Smalltalk Config Maps/Applications column and then selecting **Remove selected** from the context menu for that column. You cannot delete applications within a selected configuration map.

8. When you are ready to migrate the listed applications and configuration maps, select the **Migrate** push button. All applications and configuration maps listed in the window are migrated. Flags are set in the Smalltalk library (default name mgr50.dat) to indicate which applications and configuration maps have been migrated. The current settings for the Library Management and Ownership migration options are used for all of the applications or configuration maps that are being migrated.

During migration, the V3 to V4 Migration window is refreshed with status information in the Migration Status column. When all the applications and configuration maps have been successfully migrated, the Migration Status column shows a status of **Migrated** for each application and configuration map. The status information is also written to the status log. If an application or configuration map shows a status of **Migration Error...check status log**, you can check the status log to see which step in the migration process was the last to complete successfully for that application.

Working with the status log

The V3 to V4 Migration Tool stores a record of each step in the migration process in a status log. To view the status log, from the V3 to V4 Migration window, select the **View Status Log** push button.

Figure 23 on page 186 shows the Migration Status Log window.

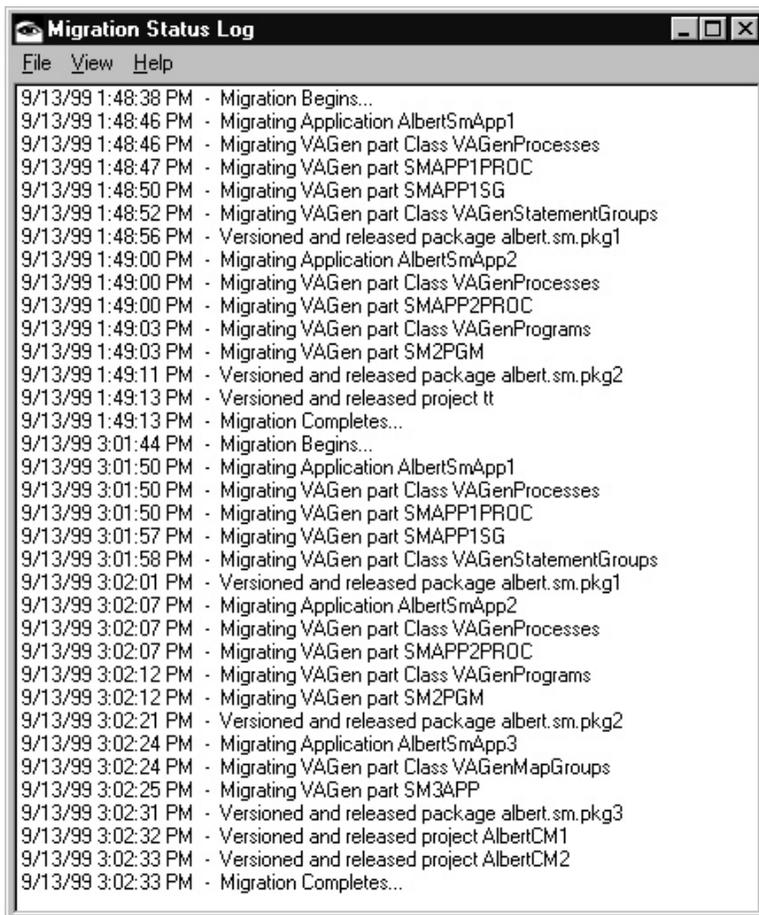


Figure 23. Migration Status Log window

The status log is stored in the file **mgstatus.log**. This log is cumulative. The information in this log is only deleted when you select **File->Clear** from the menu in the Status Log window.

If you have hundreds of applications and configuration maps to migrate, the status log can become quite large. At some point, you might want to clear the status log of the information from previous migration operations before you begin migrating the next group of applications and configuration maps.

Before clearing the status log, however, you might want to print it (by selecting **File->Print**) or copy it to another file (by selecting **File->Save as**). When you use **Save as** to save the status log to another file, if that other file exists, a prompt appears asking if you want to overwrite the file or append to it.

To clear the status log, select **File->Clear** from the menu in the Status Log window.

So that you do not have to search the entire, cumulative log to find information of interest to you, several filtering options are available that allow you to view selected portions of the status log:

All	Select this menu choice to view the entire status log contents.
Applications	Select this menu choice to view only the information about applications that have been migrated.
Errors	Select this menu choice to view only the error messages in the log.
Information	Select this menu choice to view only the information messages in the log.
Warnings	Select this menu choice to view only the warning messages in the log.
Search for string	Select this menu choice to search for a text string in the status log. Enter your search string in the dialog that is displayed after you select this option. The search string is case-sensitive.

For more information about the error, information, and warning messages displayed in the status log, see the *VisualAge Generator Messages and Problem Determination Guide*.

Resetting the V3 to V4 Migration window

The V3 to V4 Migration Tool can keep information in the V3 to V4 Migration window on all completed migration operations. As each new migration is started, the V3 to V4 Migration Tool automatically appends information to the existing contents of the window.

There might be times, however, when you do not want information on already migrated code to remain in the V3 to V4 Migration window while you are migrating the next group of applications and configuration maps. For example, you might have migrated several groups of applications and configuration maps that share common code among them but that do not contain any code shared by your remaining unmigrated applications and configuration maps. In this case, the information on past migrations is not needed for the next migration operation. To remove the unneeded information, you can reset the V3 to V4 Migration window before migrating the next group.

When you reset the V3 to V4 Migration window, all information about past migration operations is deleted from the window. To reset the window, select the **Reset** push button. When you select **Reset**, the information on past migrations is still stored in the migration status log, **mgstatus.log**, and both the **Select Config Maps** and **Select Applications** buttons are enabled again.

Resetting Migration Status Information

The V3 to V4 Migration Tool sets flags in the Smalltalk library (default name mgr50.dat) to record the migration status of configuration maps and application. There might be times when you want to reset this status. For example, after completing a pilot migration, you might want to reset the migration status before doing the final migration. To reset the status, from the V3 to V4 Migration window, select the configuration maps and applications that you want to reset. Then press mouse button 2 and select **Mark Not Migrated**. The flags in the Smalltalk library (default name mgr50.dat) are reset to indicate that the selected configuration maps and applications have not been migrated.

Note: Selecting **Mark Not Migrated** does not delete the migrated parts in the Smalltalk library (default name mgr50.dat).

Chapter 20. Using VAGen Import to migrate VAGen 3.x non-GUI code to Smalltalk

When you use VAGen Import instead of the V3 to V4 Migration Tool to migrate 3.x code to VisualAge Generator 4.0, the following restrictions apply:

- You can only migrate non-GUI code using VAGen Import.
- All configuration map information is lost. You will need to recreate configuration maps in VisualAge Generator 4.0.
- Existing ownership information is lost. Ownership is set to the userid of whichever user creates the applications and imports the parts.
- The steps in this section are for applications. If you used subapplications, you will need to create one .esf file for each subapplication that you want to migrate and then manually create each subapplication in VisualAge Generator 4.0.
- Do not forget to migrate prerequisite applications and subapplications that might not be loaded into your VisualAge Generator 3.x image based on the configuration expressions used for the parent application.

To migrate VisualAge Generator 3.x non-GUI code to VisualAge Generator 4.0 on the Smalltalk platform, use these steps:

1. Start VisualAge Generator 3.x on Smalltalk.
2. Export your existing applications to external source format files. You must use your existing VisualAge Generator 3.x product to create these .esf files.

From the VisualAge Organizer window in VisualAge Generator 3.x, select **Applications->Import/Export->VAGen Export** to create the external source format files. Export one external source format file for each existing ENVY application.

Notes:

- a. If you are migrating from VisualAge Generator 3.x on an OS/2 development platform to VisualAge Generator 4.0 on a Windows NT development platform, see section "Migrating from OS/2 to Windows NT" on page 174 for information on code page conversions.
 - b. You should not modify the export files.
3. Start VisualAge Generator 4.0 on Smalltalk.
 4. In VisualAge Generator 4.0, create a VisualAge Smalltalk application using these steps:
 - a. From the VisualAge Organizer, select **Applications->New**.

- b. In the New Application window, enter a name for the application and select **OK**. You will use this new application to receive the contents of one of the external source format files that you want to import into VisualAge Generator 4.0.

Create one ENVY application for each .esf file that you created in step 2. For more information about creating applications, see the *VisualAge Generator User's Guide*.

5. Using VisualAge Generator 4.0, from the VisualAge Organizer window, select **VAGen Parts->Parts Browser** to open the VAGen Parts Browser.
6. From the VAGen Parts Browser window, select **Parts->Import/Export->VAGen Import**.
7. In the VAGen Import File Selection window, select one external source format (*.esf) file that you want to import and then select the **Open** button.
8. In the VAGen Import window, check to see if there are any parts listed in the Parts with errors list box. If there are, you need to debug the errors before those parts can be imported with the rest of your application. When there are no parts with errors, proceed to the next step.
9. In the VAGen Import window, specify the name of your application in the Target application field. Move all parts to be imported from the Available parts list to the Selected parts list. Then select the **Import** push button. The selected parts are imported into the application you specified.
10. In the VAGen Import window, select the **New File** push button and then repeat steps 7, 8, and 9 for each external source format file you need to import. When you have finished importing your .esf files to VisualAge Generator 4.0, select the **Cancel** push button in the VAGen Import window to return to the VAGen Parts Browser window.
11. Version and release all classes and applications you have imported.
12. Save your VisualAge Generator 4.0 image.
13. See "Chapter 21. Completing the ENVY setup on Smalltalk" on page 191 and "Chapter 22. Completing your migration on Smalltalk" on page 199 for information about additional steps you might need to take.

Chapter 21. Completing the ENVY setup on Smalltalk

The following sections describe specific tasks you might need to perform after running the V3 to V4 Migration Tool. These tasks include:

- “Versioning and releasing a view or a VAGen part class”
- “Versioning an application” on page 192
- “Creating a configuration map” on page 193
- “Adding a required map to a configuration map” on page 194
- “Versioning a configuration map” on page 195
- “Changing the manager of a configuration map” on page 195
- “Assigning ownership of a VAGen part class” on page 196
- “Adding group members” on page 196
- “Changing the ownership of a VAGen part class” on page 197
- “Changing the manager of an application” on page 197

Versioning and releasing a view or a VAGen part class

After migration, you should version your views and VAGen part classes to provide a base line that reflects the level of code you migrated to VisualAge Generator 4.0. If you selected the **Auto version and release** check box on the V3 to V4 Migration Tool’s Migration Options window, all the VAGen part classes were automatically versioned and released for you during the migration process. If you did not migrate with this option, however, you now need to version and release the classes.

The steps below describe how to version and release views and VAGen part classes for an application.

Note: You can version and release a view or VAGen part class in two separate steps as described below or in a single step by selecting **Version/Release Owned**. Multiple views and VAGen part classes within a single application can be versioned or released at the same time.

Action

From the VisualAge Organizer window:

- In the **Applications** pane, select an application.
- In the **Parts** pane, select all the VAGen part classes. These are the classes that start with *VAGen* (for example, *VAGenRecords*).

Description

Note: In the **Parts** pane, VAGen part classes that need to be versioned appear in the format:

>VAGenRecords (06/03/99 10:15:30 AM)

Action	Description
<p>In the Parts pane, press mouse button 2 and select Version → and then one of the following:</p> <ul style="list-style-type: none"> • Name Each to specify a different version name for each of the classes you are versioning. • One Name to specify the same version name for all of the classes you are versioning. Use this option if you are migrating parts built with VisualAge Generator Templates or BW*Wizard or the business logic for those parts. • Use Defaults to use the ENVY-determined defaults for each of the classes you are versioning. The default might be different for each class. 	<p>The Parts pane in the VisualAge Organizer window is refreshed and shows the version number for each VAGen part class.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. In the Parts pane, VAGen part classes that have been versioned but not released appear in the format: <pre>>VAGenRecords 1.0</pre> 2. You must be the developer of a class to version the class.
<p>In the Parts pane, press mouse button 2 and select Release → Current Version.</p>	<p>The Parts pane in the VisualAge Organizer window is refreshed and shows the version number for each VAGen part class.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. In the Parts pane, VAGen part classes that have been versioned and released appear in the format: <pre>VAGenRecords 1.0</pre> 2. You must be the owner of a class to release the class.

Versioning an application

After migration, you should version your applications to provide a base line that reflects the level of code you migrated to VisualAge Generator 4.0. If you selected the **Auto version and release** check box on the V3 to V4 Migration Tool's Migration Options window, all the applications were automatically versioned and released for you during the migration process. If you did not migrate with this option, however, you now need to version and release the applications.

The steps below describe how to version and release an application.

Action	Description
<p>From the VisualAge Organizer window, in the Applications pane, select one or more applications to be versioned.</p>	<p>Note: In the Applications pane, applications that need to be versioned appear in the format:</p> <pre>TrbCommonDataApp (06/03/97 10:15:30 AM)</pre>

Action	Description
<p>In the Applications pane, press mouse button 2 and select Version → and then one of the following:</p> <ul style="list-style-type: none"> • Name Each to specify a different version name for each of the applications you are versioning. • One Name to specify the same version name for all of the applications you are versioning. Use this option if you are migrating parts built with VisualAge Generator Templates or BW*Wizard or the business logic for those parts. • Use Defaults to use the ENVY-determined defaults for each of the applications you are versioning. The default might be different for each application. 	<p>The Applications pane in the VisualAge Organizer window is refreshed and shows the version number for each application.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. In the Applications pane, applications that have been versioned but not released appear in the format: TrbCommonDataApp 1.0 2. You must be the manager of an application to version the application.

Creating a configuration map

A configuration map allows you to define a group of application editions that should all be loaded together into your image. For example, you might have a configuration map that defines the application versions that are currently in your production system.

If you used the V3 to V4 Migration Tool to migrate your 3.x configuration maps to VisualAge Generator 4.0, all your configuration maps were preserved for you during migration. The relationships between configuration maps (such as required maps) were also preserved. However, you might want to reorganize some of your code after migration to VisualAge Generator 4.0

This section describes how to create a configuration map, release applications into the configuration map, and specify any required maps for the configuration map.

Action	Description
<p>From the VisualAge Organizer window, select Tools and then Configuration Maps.</p>	<p>The Configuration Maps Browser is displayed.</p>
<p>Select Names and then Create.</p>	<p>An Information Required window is displayed prompting you for the name of the configuration map.</p>
<p>Type the name of the configuration map and select the OK push button.</p>	<p>The Configuration Maps Browser is displayed, with the name of the new configuration map included in the list. The new configuration map is highlighted.</p>

In the **Description** pane in the lower right corner, type a description of the configuration map.

Action	Description
In the Applications pane, press mouse button 2 and select Add .	A Selection Required window is displayed listing the names of existing applications.
<p>In the Selection Required window:</p> <ul style="list-style-type: none"> • In the Names pane, select an application to add to the configuration map. • In the Editions pane, select the edition of the application. • Select the >> push button to add the application edition to the Selected Editions pane. 	<p>The Configuration Maps Browser is displayed, with the applications listed in the Applications pane.</p> <p>Note: You must be the manager of a configuration map to add applications to the configuration map.</p>
Repeat this process until all the needed applications are included in the Selected Editions pane.	
Select the OK push button.	

Adding a required map to a configuration map

You can define a hierarchy of configuration maps by specifying required maps. For example, in the configuration map for a subsystem, you might want to specify the configuration map for common code as a required map.

Action	Description
From the Configuration Maps Browser, in the Config. Expressions pane, press mouse button 2 and then select Add .	<p>An Information Required window is displayed prompting you to specify an expression.</p> <p>Note: You must be the manager of a configuration map to add a configuration expression.</p>
Select the OK push button to accept the default value of true for the configuration expression.	<p>The Configuration Maps Browser is displayed, with true listed in the Config. Expressions pane.</p> <p>Specifying true means that the configuration maps listed in the Required Maps pane will always be loaded before loading the applications for this configuration map.</p>
From the Required Maps pane, press mouse button 2, and then select Add → As First .	<p>A Selection Required window is displayed prompting you for a configuration map and its edition.</p> <p>Note: You must be the manager of a configuration map to add a required map.</p>

Action	Description
Select the prerequisite map and its edition and then select the OK push button.	The Configuration Maps Browser is displayed, with the configuration map listed in the Required Maps pane.

Versioning a configuration map

After migration, you should version your configuration maps to provide a base line that reflects the level of code you migrated to VisualAge Generator 4.0. If you selected the **Auto version and release** check box on the V3 to V4 Migration Tool's Migration Options window, all the configuration maps were automatically versioned for you during the migration process. If you did not migrate with this option, however, you now need to version the configuration maps.

The steps below describe how to version a configuration map.

Action	Description
From the Configuration Maps Browser , select the following: <ul style="list-style-type: none"> • In the Names pane, select the name of the configuration map. • In the Editions and Versions pane, select the edition you want to version. • In the Editions and Versions pane, press mouse button 2 and then select Version. 	An Information Required window is displayed prompting you to specify a version number. Note: You must be the manager of a configuration map to version the configuration map.
In the Information Required window, select the OK push button to use the default version of 1.0.	The Configuration Maps Browser is displayed with the version showing in the Editions and Versions pane.

Changing the manager of a configuration map

Each configuration map can have a different manager. Follow these steps to change the manager of a configuration map.

Action	Description
From the Configuration Maps Browser, in the Editions and Versions pane, press mouse button 2 and then select Change Manager .	A Selection Required window is displayed showing the current group members for the configuration map. The current manager is highlighted. Note: You must be the manager of a configuration map to change the manager of that configuration map.
Select the new manager and select the OK push button.	The Configuration Maps Browser is displayed.

Testing a configuration map

You need to test that any new configuration maps you create will load successfully. To do this, you can unload the applications and then reload them using the configuration maps as described below. Alternatively, test the configuration maps by trying to load them into a clean image.

Action	Description
From the VisualAge Organizer window, select all the applications you created.	The VisualAge Organizer window is refreshed and no longer shows the applications you created.
Select Applications and then Unload .	
From the Configuration Maps Browser, select a configuration map and then select the current version.	The applications in the configuration map are loaded and now appear in the VisualAge Organizer window.
From the Editions and Versions pane, press mouse button 2 and then select Load With Required Maps (or Load if there are no required maps for this configuration map).	
Check the System Transcript window for any error messages.	

Assigning ownership of a VAGen part class

Each VAGen part class within an application can have a different owner. Although developers can version a VAGen part class, the class owner controls the release of VAGen part classes into the application.

The following sections describe how to add members to the authorized group for an application and how to change the owner of a VAGen part class accessed by that application group.

Adding group members

To change the owner of the class, the new owner must be a member of the group. Group members must be defined for one application at a time.

Action	Description
From the VisualAge Organizer window, select the application for which group members are to be added.	A window showing all defined users and the current group members for the application is displayed. The current manager is indicated by a >.
Select Applications and then Group Members .	Note: You must be the manager of an application to add group members for that application.
From the Users pane, select a user and select the >> push button to move the user to the Group Members pane.	
Repeat this step to add any additional users to the group.	

Action	Description
When all new users have been moved to the Group Members pane, select the OK push button.	The VisualAge Organizer window is displayed.

Changing the ownership of a VAGen part class

The same owner can be assigned for multiple classes within a single application at the same time. During migration, it is easiest to assign the same owner to all the VAGen part classes within an application. The owner of these classes might also (most likely) be the manager of the containing application.

During the V3 to V4 Migration Tool processing, ownership of all the classes was automatically assigned for you. However, you can use the steps below to change the ownership of any VAGen part class.

Action	Description
From the VisualAge Organizer window: <ul style="list-style-type: none"> • In the Applications pane, select the application for which VAGen part class owners are to be assigned. • In the Parts pane, select the VAGen part classes for which the same owner is to be assigned. 	A Selection Required window is displayed showing the current group members for the application. The current owner is highlighted. Note: You must be the manager of an application or the owner of a class to change the owner of that VAGen part class.
Select Parts and then Owner → Change Owner .	
Select the new owner and select the OK push button.	The VisualAge Organizer window is displayed.

Changing the manager of an application

Each application can have a different manager. The manager of an application can create an edition of the application and can version the application.

To change the manager of an application, the new manager must be a member of the group for that application. See “Adding group members” on page 196 for information on adding users to a group.

Action	Description
From the VisualAge Organizer window, in the Applications pane, select the application for which the manager is to be changed.	A Selection Required window is displayed showing the current group members for the application. The current manager is highlighted.
Select Applications and then Manager → Change Manager .	Note: You must be the manager of an application to change the manager of that application.

Action**Description**

Select the new manager and select the **OK** push button.

The VisualAge Organizer window is displayed.

Chapter 22. Completing your migration on Smalltalk

The following sections describe specific tasks you might need to perform to complete your migration to VisualAge Generator 4.0. These tasks include:

- “Defining control information”
- “Generating programs”
- “Importing work-in-progress”
- “Recreating ITF resource association information” on page 200

Defining control information

Control information that is needed for test and generation must be stored in ENVY applications. This control information consists of:

- Generation options
- Linkage table
- Resource associations
- Bind control information
- Link edit information

When you use the V3 to V4 Migration Tool to migrate your 3.x applications, you can migrate control information just as you would migrate any other ENVY applications.

Generating programs

After you finish migrating a group of Smalltalk applications to VisualAge Generator 4.0, you might want to generate the programs to help ensure that you have migrated the correct version of your code.

If you are migrating from VisualAge Generator 3.0, any programs for the C++ target environment **must** be regenerated.

Importing work-in-progress

Suppose you have already migrated and versioned a development system. Then, three weeks after migration, you discover 50 4GL parts have been changed on the VisualAge Generator 3.x development system. The 4GL parts are contained in several applications. You can use Smalltalk export / import and the V3 to V4 Migration Tool to migrate the new versions of the changed applications. Be sure to set your migration options to the same settings as in your original migration. The V3 to V4 Migration Tool will do the following:

- Load the affected applications.
- Create new editions of all 4GL parts from the corresponding parts in the modified applications.

The advantage of using the V3 to V4 Migration Tool is that you only need to identify the applications that must be re-migrated. Another advantage is that migration of this version of the applications is logged in the status log, **mgstatus.log**. The disadvantage is that all the 4GL parts in the changed applications are re-migrated and result in new editions in the Smalltalk library.

Alternatively, you can export an external source format file that contains only the 4GL parts that have been changed. You can then use **VAGen Import** to migrate the parts. See “Importing work-in-progress” on page 341 for information on how to use VAGen Import to migrate your work-in-progress. Using **VAGen Import** requires you to load the affected applications into your image. The advantage of using **VAGen Import** is that you only need to migrate the 4GL parts that were changed, rather than all the 4GL parts in the affected applications.

Recreating ITF resource association information

If you use serial, indexed or relative files in the Interactive Test Facility, you also use resource association information to point to the files. ITF resource association information is not stored as a resource association part. You need to recreate the ITF resource association information for VisualAge Generator 4.0 on Smalltalk.

Converting an RTABLE to a Linkage Table

If you have been using the VisualAge Generator middleware RTABLE for communications routing, the RTABLE entries must be moved from the RTABLE to a linkage table. The following example shows a mapping of RTABLE entries to linkage table entries:

```
RTABLE
app1 - - - - - lu2 LU2C - 1
app2 - - - - - lu2 LU2C - 1
app3 - - - - - lu2 LU2C - 1
app4 - - - - - lu2 LU2B - 1
$ANY - - - - - lu2 LU2K - 1

LINKAGE TABLE
:calllink applname=app1 linktype=remote remotecomtype=LU2
serverid=LU2C.
:calllink applname=app2 linktype=remote remotecomtype=LU2
serverid=LU2C.
:calllink applname=app3 linktype=remote remotecomtype=LU2
```

```
serverid=LU2C.  
:calllink applname=app4 linktype=remote remotecomtype=LU2  
serverid=LU2B.  
:calllink applname=* linktype=remote remotecomtype=LU2  
serverid=LU2K.
```

Part 4. Migrating from VAGen 2.x or Cross System Product to VAGen 4.0 on Smalltalk

Chapter 23. Comparing MSLs and ENVY on VAGen 4.0 on Smalltalk	207
ENVY characteristics.	207
Comparison of MSLs and ENVY.	209
Member types	209
Storing members	210
Storing control information	210
MSL concatenation	211
Functional organization.	212
Member associations.	212
Chapter 24. General migration considerations for VAGen 2.x and Cross System Product to Smalltalk	215
Migration paths	215
Automatic conversions during 4.0 migration	216
Resolving duplicate member names.	217
Establishing naming conventions.	217
Organizing your code for ENVY.	219
Assigning ownership	220
Using subapplications	221
Storing control information	223
Multiple control applications	223
Single control application	224
Generation options	225
Using configuration maps	225
Migrating GUIs	226
Conversion of GUIs to VisualAge Generator 4.0	228
Tasks that take place during GUI conversion	229
Obsolete parts maintained for compatibility	232
Tasks you must do after migrating GUIs	234
Dual maintenance.	235
Migrating from Cross System Product	236
Migrating from OS/2 to Windows NT	237
Using the migration log file	237

Chapter 25. Pre-migration checklist	239
--	-----

Chapter 26. The MSL Migration Assistance Tool on Smalltalk	241
Overview of using the MSL Migration Assistance Tool	242
Building MSL directories	243
Selecting MSLs - using the MSL Library Selection window	244
Selecting parts for a part list - using the Part List Selection Criteria View window	245
Selecting parts to move to ENVY - using the MSL Migration Part List window	246
Special columns in the MSL Migration Part List window	248
Other MSL Migration Part List functions.	250
Working in the sandbox - using the VG Part Prerequisites View window	250
Identifying common code	250
Identifying missing parts	251
Other sandbox functions	251
Resetting the MSL Migration Assistance Tool sandbox	252

Chapter 27. Migrating production and work-in-progress MSLs to VAGen 4.0 on Smalltalk	255
Production MSLs	256
Techniques for moving parts to the sandbox	257
All data items, records, and tables are used	257
All records and tables are used	258
Some records and tables might not be used	259
Considerations for the migration techniques	259
Work-in-progress MSLs	261
Using VAGen Import	263
Using the MSL Migration Assistance Tool	264

Chapter 28. VAGen on Smalltalk case studies based on various MSL structures.	265
---	-----

Understanding the diagrams and terminology	265	Resetting the sandbox from ENVY	302
MSL structure diagrams.	265	Selecting your MSLs	304
Advance command in VisualAge Generator	266	Selecting and migrating VAGen parts	305
Multiple subsystems with no duplicates	267	Creating a new application.	308
Recommendations	267	Moving a VAGen part between applications	308
Multiple subsystems with controlled duplicates	269	Controlling the creation of ApplicationNodes	310
Recommendations	269	Renaming an application	311
Separate production MSLs for each developer	272	Collapsing an application	311
Recommendations	273	Handling Duplicates	312
Creating a core common set of applications.	273	Controlled Duplicates	312
Creating stand-alone subsystems.	275	Unintended Duplicates	313
MSLs that contain unintended duplicates	277	Duplicates for business logic	315
Recommendations	278	Finding the application in which a part is located	316
MSLs containing code from VisualAge Generator Templates or BW*Wizard.	278	Listing missing (not found) parts	317
Recommendations	280	Handling missing (not found) parts.	318
Special considerations for VisualAge Generator Templates.	282	Checking relationships among applications	319
Complete set of MSLs for production and deltas for test	282	Determining which programs are referenced	319
Recommendations	284	Determining the parts that are referenced	320
Complete sets of MSLs for test and production	285	Checking consistency of applications	321
Recommendations	287	Updating the list of required applications	321
MSLs for marketing or other demonstrations	289	Changing the list of required applications	322
Recommendations	290	Normalizing the list of required applications.	322
Using a single ENVY application	290	Deleting an ApplicationNode	323
Using multiple ENVY applications	291	Deleting an application	324
		Deleting one application	324
		Deleting all applications	324
		Committing to ENVY	325
Chapter 29. Running the MSL Migration Assistance Tool on Smalltalk	293	Chapter 30. Using VAGen Import to migrate VAGen 2.x and Cross System Product code to Smalltalk	329
Starting VisualAge Generator	293	Chapter 31. Completing the ENVY setup on Smalltalk	331
Creating users and setting the current user	294	Versioning and releasing a view or a VAGen part class	331
Loading a feature	295	Versioning an application	332
Collecting your source code	296	Creating a configuration map.	333
From Cross System Product	296	Adding a required map to a configuration map	334
From VisualAge Generator with TeamConnection and no MSLs	296	Versioning a configuration map	335
From VisualAge Generator MSLs	297	Changing the manager of a configuration map	335
Handling code page changes	297	Testing a configuration map	335
Using the HPTRULES.NLS file	297	Assigning ownership of a VAGen part class	336
Changing from OS/2 to Windows NT	298	Adding group members	336
Starting the MSL Migration Assistance Tool	300		
Building MSL directories	300		

Changing the ownership of a VAGen part class	336
Changing the manager of an application	337

Chapter 32. Completing your migration on Smalltalk	339
Defining control information	339
Generating programs and packaging views	340
Importing work-in-progress	341
Migrating VSAM files	343
Converting an RTABLE to a Linkage Table	344

Chapter 33. Hints and tips on Smalltalk	345
System Transcript Window.	345
VisualAge Organizer window.	345
VAGen Parts Browser window	345
Refreshing the MSL Migration Assistance Tool	346

Chapter 23. Comparing MSLs and ENVY on VAGen 4.0 on Smalltalk

In both Cross System Product and releases of VisualAge Generator prior to 3.0, code was written in small pieces called members. Members were stored in Member Specification Libraries (MSLs). In VisualAge Generator 3.0 and later, the code must be stored in ENVY, a library management system.

If you plan to use code from Cross System Product and previous releases of VisualAge Generator, you must migrate the code from the MSLs to ENVY. This chapter explains the following:

- “ENVY characteristics”
- “Comparison of MSLs and ENVY” on page 209

ENVY characteristics

To provide interoperability with VisualAge Smalltalk, VisualAge Generator 4.0 shares the VisualAge Smalltalk library management system. This library management system is called ENVY.

There are new terms that are important for the ENVY environment. The following terms are new for VisualAge Generator 4.0:

New Term

VAGen part class

Relationship to MSL Concepts

Each 4GL member type (all member types except GUIs) becomes a **VAGen part class**. A VAGen part class is an **extension of a class** in Smalltalk. The VAGen part classes appear in the **Parts** pane of the VisualAge Organizer window. The VAGen part classes created for the member types are prefixed by *VAGen* (for example, *VAGenRecords*).

There are five additional VAGen part classes that are used to contain control information that was stored outside the MSL in previous releases of VisualAge Generator. These VAGen part classes are for linkage table, resource association, generation options, bind control, and linkage editor information.

VAGen part

Each 4GL member is now stored as a **VAGen part**. A VAGen part is associated with a

Smalltalk **method** in an extension of its VAGen part class. The VAGen parts appear in the **VAGen Parts** pane of the VisualAge Organizer window and in the VAGen Parts Browser window.

View

Each GUI is now stored as a **view**, which is a **visual part**. A view is a **class** in Smalltalk. The views appear in the **Parts** pane of the VisualAge Organizer window. Views do not appear on the VAGen Parts Browser in ENVY.

Program

The application member type has been changed to **program** to distinguish it from an ENVY application.

ENVY Application

An ENVY application is a group of classes and methods that are closely related in function. An ENVY application can include VAGen part classes and VAGen parts. An ENVY application is also called an **application**.

Configuration Map

A **configuration map** is a group of application editions that should be loaded together into a developer's image.

The following ENVY concepts are new for VisualAge Generator 3.0 and later:

Concept

Description

Functional Organization

ENVY enables you to group parts into applications. These applications can (and should) be organized along functional lines.

Ownership

Each configuration map and each application has an assigned manager who is responsible for the integrity of the code that is placed in the configuration map or application.

Each part (class) or VAGen part class has an assigned owner who is responsible for the integrity of the code that is placed in the class. For 4GL parts, this means that the owner of the class *VAGenRecords* within application XYZ is responsible for the integrity of **all** record definitions stored as part of application XYZ. Because each GUI becomes a separate view (visual part), each GUI within application XYZ

can have a different owner. 4GL parts used within the GUI become VAGen parts.

Note: In Cross System Product and VisualAge Generator 2.x or earlier, the closest concept to ownership was write-protecting the staging, test, or production MSLs and only giving a team leader the authority to advance members into these MSLs.

Edition	Each change that is made to a 4GL VAGen part results in a new edition of the VAGen part being stored in the ENVY library manager. Editions of parts, applications, and configuration maps are also stored in the ENVY library manager.
Version	Editions can be frozen to prevent further changes to that level of code. The frozen edition is called a version . After a part, application, or configuration map is versioned, the only way to make changes is to open a new edition.
Image	An image is the developer's current view of the ENVY library manager. It contains the version or edition of the configuration maps, applications, and parts that the developer wants to work on. Only one copy of a VAGen part can be loaded into the image at one time.

Comparison of MSLs and ENVY

This section describes how concepts you are familiar with for MSLs relate to concepts in the ENVY library manager.

Member types

In Cross System Product and releases of VisualAge Generator prior to 3.0, there was one member type for each type of code that could be written.

With VisualAge Generator 3.0 or later, each 4GL member type is a VAGen part class that is prefixed with *VAGen* (for example, *VAGenRecords*). For GUIs, there is no corresponding VAGen part class because each GUI becomes a separate class.

Table 20 on page 361 shows the correspondence between member types and VAGen part classes.

Storing members

In releases of VisualAge Generator prior to 3.0, an MSL was an OS/2 directory and each member was a file within the directory. In Cross System Product, an MSL was a VSAM file and each member was stored as records within the file.

With VisualAge Generator 3.0 or later, all information is stored in the ENVY library manager. Each 4GL member is a VAGen part and is associated with a Smalltalk method. Each GUI is a view (visual part) and is a Smalltalk class.

Storing control information

For Cross System Product and releases of VisualAge Generator prior to 3.0, most control information related to test and generation was stored outside the MSL. This control information included:

- Generation options that indicate how an application is to be generated. For example, generation options control whether working storage records are to be initialized, what high-level qualifier on the host is to be used for preparation, and what linkage table is to be used.

For Cross System Product, only COBOL generation used generation options and these were stored in separate files outside the MSL. For releases of VisualAge Generator prior to 3.0, COBOL generation options were stored in separate files.

- Linkage table that indicates how a CALL, DXFR, or XFER is to be implemented. For the CICS target environment, the linkage table is also used to indicate whether a VSAM or transient data queue is to be accessed locally or remotely. For example, the linkage table might specify that a CALL to application XYZ in the MVS CICS target environment is to be implemented as a remote call passing data in the CICS COMMAREA.

For Cross System Product prior to 4.1, there was no linkage table. For Cross System Product 4.1 and releases of VisualAge Generator prior to 3.0, the linkage table was in a separate file.

- Resource association file that indicates for a specific file how it is to be implemented in a specific target environment. For example, a serial file in the MVS CICS target environment might be implemented as a VSAM file, as a transient data queue, as a temporary storage queue, or as a CICS spool file.

For Cross System Product, resource association information was stored in the MSL. For releases of VisualAge Generator prior to 3.0, resource association information was stored in a resource association file outside the MSL.

- Bind control commands that provide information needed for binding the DB2 application plan on an MVS system.

For Cross System Product/AE, bind control information was not used. Cross System Product COBOL generation used bind control commands and

these were stored in separate files outside the MSL. For releases of VisualAge Generator prior to 3.0, bind control commands were stored in separate files.

- Linkage editor control statements that provide linkage editor information for MVS, VSE, and VM systems.

For Cross System Product/AE, linkage editor control statements were not used. Cross System Product COBOL generation used linkage editor control statements and these were stored in separate files outside the MSL. For releases of VisualAge Generator prior to 3.0, linkage editor control statements were stored in separate files.

With VisualAge Generator 3.0 and later, generation options files, linkage tables, resource association files, bind control commands, and linkage editor control statement files are methods within VAGen part classes in ENVY. Thus all the data required for test and generation is contained in a single library management system.

Table 15 shows the correspondence between the types of control information and VAGen part classes.

Table 15. Control Information and VAGen Part Classes

Control Information	VAGen Part Class
Generation Options	VAGenOptions
Linkage Table	VAGenLinkages
Resource Associations	VAGenResources
Bind Control Commands	VAGenBindControls
Linkage Editor Control Statements	VAGenLinkEdits

MSL concatenation

In Cross System Product and VisualAge Generator 2.x and earlier, MSLs could be concatenated. Specifying the MSL concatenation sequence was the way in which you specified where to look for the members needed by your applications. When MSLs were concatenated, for test and generation, only the first found member with a given name was used. For viewing, members from MSLs other than where the first found member was located could be referenced. If changes were made to a member in the MSL concatenation sequence, the changed member was stored in the read/write MSL (first MSL in the concatenation sequence).

With VisualAge Generator 4.0, there is no concept similar to the first found member in an MSL concatenation. All editions are available in the ENVY library manager. However, only one edition of a part can be loaded into your

image at a time. Browsers are available to compare editions within the ENVY library manager before loading them into your image to determine which one is the required level of code.

You can use configuration maps to group the code for a particular level. For example, you might have a configuration map for production that indicates the version of each application that is in production. Specifying required maps for a configuration map is similar to specifying the MSL concatenation sequence. The required maps provide a way for you to ensure that all the parts needed to run a particular group of programs are loaded into your image together. Therefore, required maps resolve the problem of first found parts. For more information, see the following:

- “Using configuration maps” on page 225
- “Creating a configuration map” on page 333
- “Adding a required map to a configuration map” on page 334

Functional organization

In Cross System Product and VisualAge Generator 2.x and earlier, members were grouped together into MSLs. Generally, an MSL contained all the members for a particular subsystem. Because the MSL was the only method for grouping members by function, the functions tended to be quite large. In Cross System Product the number of MSLs in a concatenation sequence was limited to 6. This also contributed to having a large number of members in each MSL.

With VisualAge Generator 4.0, the configuration map and the ENVY application provide a two-level capability for grouping parts. The configuration map is the higher level of organization and more closely resembles an MSL in terms of the number of parts. ENVY applications enable you to organize your parts into smaller groups than was reasonable to do with MSLs. This provides more capabilities in terms of controlling access to the parts, finding a part, and limiting the number of parts displayed in the VAGen Parts Browser window.

Member associations

For Cross System Product and VisualAge Generator 2.x and earlier, members could have associates — other members that were referenced within the member. With VisualAge Generator 4.0, the same associations between 4GL VAGen parts still exist. For example:

Member Type	Member Types of Possible Associates
Data Item	None
Process	None
Statement Group	None

Record	Any global data items
Table	Any global data items
PSB	Records for the segment records, any global data items used by those records, and any item named as a secondary index field in the PSB
Map	Any other maps in the same map group, any statement group or table used as an edit routine, and any global data item in the table
Map Group	Any maps in the map group and their associates
Application	Map groups, maps, records, tables, PSB, processes, statement groups, and their associates. Any global data item referenced directly or indirectly was included. For example, a data item used as a called parameter was included as an associate of the application. Only maps that were actually used by the application were included in the associates.

Notes:

1. References to processes and statement groups in the above list represent the MSL terminology. These VAGen part types have been merged into the new VAGen function part type.
2. References to global data items in the above list represent the MSL terminology. Global data items have been renamed to *shared data items* in VisualAge Generator 4.0.
3. References to applications in the above list represent the MSL terminology. Applications have been renamed to *programs* in VisualAge Generator 3.0 and later.

For a specific member, for example application XYZ, the only associations that could be detected were those that existed in the current MSL concatenation sequence, using the first found members.

For VisualAge Generator 4.0, the only associations that can be detected are the ones that exist within the current image.

For VisualAge Generator 2.x and earlier, the associates of GUIs were records, tables, processes, statement groups, or other embedded GUIs, and their associates. An external GUI and its associates were not included.

With VisualAge Generator 4.0, for a view, the following are associates:

- 4GL parts — records, tables, and functions and their associates.
- For any embedded view, its associates. However, the embedded view itself is not included as an associate.

Chapter 24. General migration considerations for VAGen 2.x and Cross System Product to Smalltalk

Consider the following if you are migrating from VisualAge Generator 2.x or Cross System Product to VisualAge Generator 4.0 on Smalltalk:

- “Migration paths”
- “Automatic conversions during 4.0 migration” on page 216
- “Resolving duplicate member names” on page 217
- “Establishing naming conventions” on page 217
- “Organizing your code for ENVY” on page 219
- “Assigning ownership” on page 220
- “Using subapplications” on page 221
- “Storing control information” on page 223
- “Using configuration maps” on page 225
- “Migrating GUIs” on page 226
- “Dual maintenance” on page 235
- “Migrating from Cross System Product” on page 236
- “Migrating from OS/2 to Windows NT” on page 237

Also see “Chapter 25. Pre-migration checklist” on page 239 before you begin to migrate code to VisualAge Generator 4.0.

In addition, if you are migrating from Cross System Product, it is strongly recommended that you read *Migrating Cross System Product Applications to VisualAge Generator* and “Appendix D. Notes on Cross System Product migrations” on page 395 before you migrate.

Migration paths

Depending on your current platform and whether you want to migrate to Smalltalk, Java, or both, different migration paths are available and different considerations apply.

Table 16 on page 216 gives a brief overview of the migration options available for the Smalltalk platform. See the referenced chapters for step-by-step procedures for each migration option.

Table 16. Migration Options

Migrating from	Tools to Use	Details on Using the Tools
VisualAge Generator 2.x or Cross System Product (GUI and non-GUI)	<ul style="list-style-type: none"> • MSL Migration Assistance Tool • VAGen Import 	<ul style="list-style-type: none"> • “Chapter 26. The MSL Migration Assistance Tool on Smalltalk” on page 241 • “Chapter 29. Running the MSL Migration Assistance Tool on Smalltalk” on page 293 • “Chapter 30. Using VAGen Import to migrate VAGen 2.x and Cross System Product code to Smalltalk” on page 329

Automatic conversions during 4.0 migration

VisualAge Generator 4.0 automatically makes the following changes to your applications during migration:

- Members become VAGen parts.
- Process and statement group members are converted to function parts.
- References to processes and statement groups are converted to the new syntax requirements for functions with no parameters.
- PERFORM statements and Unconditional Branch statements are no longer supported and are migrated to Function Invocation statements.
- Subscript parentheses are changed to brackets in VisualAge Generator item names in the following places:
 - 4GL statements in functions (processes and statement groups)
 - Host variable names in SQL statements
 - Comparison value item in DL/I specifications
 - EZEDLPCB is used in a called parameter list
- Calls to EZE service routines are converted to the corresponding function invocation statement. A statement to set the value of EZEREPLY is also added before the function invocation.

In addition, the names of part classes, control information files, and VisualAge Generator palette parts are changed during migration to VisualAge Generator 4.0:

- Table 20 on page 361 shows the conversions made between member types and VAGen 4.0 part classes during 4.0 migration.

- Table 21 on page 361 shows the correspondence between control information files and VAGen 4.0 part classes.
- Table 22 on page 362 shows the changes made to VisualAge Generator palette parts names during 4.0 migration.
- Table 23 on page 362 shows the changes made to VAGen Templates part class names and repartition during 4.0 migration.

Resolving duplicate member names

Duplicate member names should be resolved (or at least understood) before starting the migration. Duplicates can arise in the following situations:

- A duplicate member was accidentally copied into the wrong MSL and was never deleted. In this case, you must determine which is the correct version of the member.
- Duplicate members that have the same name, but are different member types must be resolved. This might require renaming one of the members and changing all references to it.
- The duplicate member is in a staging MSL or test MSL and represents work-in-progress. In this case, the changed member should be loaded after the production MSL(s) are loaded and versioned within ENVY. Alternatively, these duplicate members might be ones that were created, then forgotten, and never intended for production.
- The duplicate member is due to a demonstration that needs to reflect the state of the demonstration at different times (for example, the initial state of the MSL, the MSL after some changes have been made, and the MSL after another set of changes have been made). In this case, it might be better to create a complete MSL representing the entire set of code at various times during the demonstration and make a separate ENVY application for each complete MSL.

Establishing naming conventions

You probably already have naming conventions for parts like programs, processes, records, and so on that you are migrating from Cross System Product or VisualAge Generator 2.x. You can continue to use your existing naming conventions for these parts, because VisualAge Generator 4.0 retains existing part names during migrations from VisualAge Generator 2.x and Cross System Product.

However, VisualAge Generator does change some existing part *types*, as well as making some syntax conversions. For example, process members and statement group members are both changed to the function part type. Therefore, you might want to establish a naming convention for function parts that is similar to your conventions for processes and statement groups.

See “Automatic conversions during 4.0 migration” on page 216 for a list of changes that VisualAge Generator makes to your existing code during migration.

You need to establish naming conventions for the new application and configuration map objects that you must create in Smalltalk for your migrated code. Your naming conventions should include the following:

- Your applications should have a unique prefix (possibly three characters) so that all of your applications will be grouped alphabetically in the VisualAge Organizer. Using a single prefix for your applications means that all of your applications appear together on the VisualAge Organizer. If there are several subsystems, each subsystem could have a different prefix, but this means that they might not appear together on the VisualAge Organizer.

For example, if you have an Accounting system and a Payroll system, you might choose to prefix your ENVY applications with *Acct* for the Accounting system and *Pay* for the Payroll system. However, there are also sample applications that ship with VisualAge Generator. These are prefixed with *Hpt*. If you have the applications for accounting, payroll, and the VisualAge Generator sample applications all loaded into your image at the same time, the order in which the applications appear on the VisualAge Organizer window is:

```
Acct
Hpt
Pay
```

Using prefixes that start with different letters can be an advantage, because some VisualAge windows allow you to enter one letter to quickly tab to the group of applications that start with that letter.

If you want all your applications to appear together, use a naming convention such as *XyzAcct* for the Accounting system and *XyzPay* for the Payroll system, where *Xyz* is an acronym for your company name. In this case, the order in which the applications appear on the VisualAge Organizer window is:

```
Hpt
XyzAcct
XyzPay
```

Note: There is no published list of prefixes reserved for IBM use. However, *Abt* is used by VisualAge Smalltalk and *Hpt* is used by VisualAge Generator.

- **Versions** should also have a naming convention. Using the default naming convention (1.0, 1.1, and so on) might be the best way to start.

In addition, you need to consider the following VisualAge Smalltalk naming conventions:

- The first character of each application or configuration map name must be an uppercase, alphabetic character. For the rest of each name, it is customary to capitalize the first letter of each word in the name.
- It is customary for the phrase **App** to be appended to the end of application names.
- It is customary for the phrase **Cfmap** or **Map** to be appended to the end of configuration map names.

Organizing your code for ENVY

For an overview of the differences between MSLs and ENVY, see “Chapter 23. Comparing MSLs and ENVY on VAGen 4.0 on Smalltalk” on page 207.

ENVY works best when applications are organized along functional lines. This provides several benefits:

- Helps limit the scope of a change to one or just a few applications.
- Supports the concept of ownership described in “Assigning ownership” on page 220.

Note: Performance is best when you limit an application to a maximum of 600-700 VAGen parts per VAGen part type. It is also recommended that you use no more than 50 classes per application. When considering this maximum, remember that processes and statement groups are both collapsed into the *VAGenFunctions* class.

You should divide your MSLs into functional areas, with one ENVY application for each functional area. Some examples are:

- A DBA MSL that contains data items and record definitions. This might be small enough to become a single ENVY application. However, you might also decide to organize it into smaller applications by dividing it using one or more of the following suggestions:
 - One application for the tables and data items that are shared by multiple subsystems and a separate application for each subsystem that contains the data items and records that are unique to that subsystem.
 - One application for the tables and data items used in a particular target environment.
 - One application for a group of closely related tables, where you want smaller applications than any of the previously mentioned techniques.
- An MSL containing common code might be small enough (if you only have a few common parts) to become a single ENVY application. However, you should consider organizing it along functional lines such as:
 - Error and message handling
 - Security support
 - Audit log or journal support

- Tools only used during development, not in the production system

You can also use the following techniques to assist you in organizing your parts into functional areas:

- You might have used a naming convention to help you identify parts of a system. For example, PAYWK might be the first five characters of all the parts that are unique to processing the weekly payroll. PAYS� might be the first five characters of all the parts that are unique to the salary payroll system. In this case, you could put all the parts that start with PAYWK into one ENVY application and the parts that start with PAYS� into a different ENVY application.
- For a menu system, if there are 10 items on the main menu, you might start by putting all programs that are used when the user selects option 1 into one application, all programs that are used when the user selects option 2 into another application, and so on. If you use this technique, consider called programs as well as programs that are transferred to using XFER or DXFR. This technique has the advantage of including programs that pass data among themselves into the same ENVY application. If you have multiple layers of menus, you could use this technique at the lowest level of menus.
- You might have designed and organized your applications in the past using a technique that is unique to your organization. Using the same organizing scheme when you migrate to Smalltalk would provide continuity for your development organization and reduce the time needed for skills transfer to the new development environment.

With any of these techniques, if you use the MSL Migration Assistance Tool, it can help you identify common code that will be shared by several ENVY applications and help you split this common code into a separate applications.

Assigning ownership

Each class within an ENVY application has an owner. This includes visual parts and VAGen part classes. Each ENVY application and each configuration map has a manager. The classes within an ENVY application can have different owners and the owners can be different from the manager of the application.

Your ownership strategy should reflect how development and maintenance responsibilities are divided in your company. You need to provide a backup mechanism in case an owner is unavailable.

Keep the following in mind when assigning ownership:

- In general, parts should be grouped into functional components so that ownership can be assigned for maintenance.

- If one developer typically handles an entire program, then that developer should be the owner for the corresponding ENVY application and all the classes in that ENVY application.
- Make the applications small enough that a single developer does not own the entire system (unless the single developer is the only person who ever works on that system).
- Make the applications small enough that very few people are working in the same application at the same time.
- The owner of a subapplication can be different from the owner of the application. The owner of a class within a subapplication can be different from the owner of the same class for the application.
- If a single developer handles both the view and the server program, then they can be grouped in a single ENVY application.

If one group of developers handles the views and a different group of developers handles the server programs, then it might be best to separate the views and server programs into different ENVY applications.

Using subapplications

You can use subapplications to break an application into smaller components. Subapplications should only be used for organizational purposes. They should not be used if the functions they represent might be shipped separately. For example, if an accounting system will always ship accounts receivable and accounts payable as a single system, then the use of subapplications to separate them within a single application would work. However, if the accounts receivable system might be shipped separately from accounts payable, then each system should have its own application, with a third application containing the common parts.

Notes:

1. The MSL Migration Assistance Tool does not support subapplications. Therefore, until all MSLs have been migrated to ENVY, it is best not to start using subapplications.
2. If you might eventually migrate to VisualAge Generator on Java, it would be best not to use subapplications at all, because VisualAge for Java does not support subapplications.

If you plan to use subapplications after completing migration, there are two alternatives:

1. Use the MSL Migration Assistance Tool to organize the parts into what will eventually be the subapplications and name each subapplication with a suffix such as SubAppMig. This is because ENVY does not allow you to use the same name for both a subapplication name and an application name.

The advantage of this technique is that the MSL Migration Assistance Tool can help you organize your parts into applications. However, if you use this technique, when you are ready to create the subapplications, you will need to:

- a. Create the containing application.
 - b. Determine the prerequisites for all the subapplications that will be part of this containing application and add them to the prerequisites list for the containing application. You can find the prerequisites from the VisualAge Organizer window by selecting one of the candidate subapplications and then selecting **Applications** → **Prerequisites**. The list of *Prerequisite applications* at the top of the window is the list that must be added to the prerequisites list for the containing application. Only the immediate prerequisites need to be added to the containing application. Candidate subapplications for this containing application do not need to be added to the containing application.
 - c. For each application that specifies one of the candidate subapplications as a prerequisite, open an edition of that application and change it to add the new containing application as a prerequisite. The list of *Dependent applications* at the bottom of the Prerequisites window is the list that should specify the candidate subapplication as a prerequisite. Only the immediate dependents need to have their prerequisites changed.
 - d. Create the new subapplications.
 - e. Move the parts from the old applications to the new subapplications.
 - f. For each configuration map that includes one of the candidate subapplications, open an edition of that configuration map and change it to include the containing application and remove the old candidate subapplications.
 - g. For each application that specifies one of the candidate subapplications as a prerequisite, change that application to delete the old candidate subapplication as a prerequisite.
 - h. Purge the old candidate subapplications.
2. Put all the parts for all the subapplications of a given application into that application. The disadvantage of this technique is that you cannot use the MSL Migration Assistance Tool to organize your eventual subapplications. However, when you are ready to create the subapplications, you only need to do the following:
 - a. Create the subapplications.
 - b. Move the parts from the containing application to each of the subapplications.

With this technique, the prerequisites for other applications do not need to be changed because they already use the containing application. Similarly, configuration maps do not need to be updated.

Storing control information

In VisualAge Generator 2.2, files were used to store the following control information:

- Linkage table
- Resource association file
- Generation options
- Bind commands
- Linkage editor control statements

In VisualAge Generator 4.0, the control information is stored in parts in the ENVY library manager.

You probably need to use different generation options for development, system test, acceptance test, and production. For example, you might need to specify different load libraries for the outputs of preparation or point to a different DB2 system depending on your level of testing. Similarly, you might need different resource associations, linkage editor control statements, and bind statements.

There are two alternative techniques for organizing control information:

- Use multiple control applications.
- Use a single control application.

Multiple control applications

One option is to store the control information for each level of test in a different ENVY application. The application names could use the naming convention:

```
sssControlCommon  
sssControlDev  
sssControlSysT  
sssControlAccT  
sssControlProd  
sssControlEmergency
```

where:

sss	Is the subsystem ID that is the same prefix you use for the other applications in the subsystem.
Common	Is used for control information that is common to test and production. None of the part names in Common is duplicated in Dev, SysT, AccT, Prod, or Emergency.
Dev	Is for the developers to use.
SysT	Is for systems test.
AccT	Is for acceptance test.

Prod Is for production.
Emergency Is for emergency fixes.

The parts might be named as follows:

```
sssMvsCicsGenOpts  
sssVseCicsGenOpts  
  
sssResourceAssociations  
  
sssLinkageTable  
  
pppppp.BDC    (for bind information that is unique to a program)  
pppppp.LED    (for linkage editor information that is unique to a program)
```

With this technique, you have `sssControlCommon` and only one of the other `sssControlxxxx` applications loaded into your image. During test and generation, the part name you specify for a particular type of control information is the same part name regardless of whether you are working at the development, system test, acceptance test, or production levels of code.

The configuration maps for development, system test, acceptance test, and production each include `sssControlCommon` and the `sssControlxxxx` application that corresponds to their level of testing.

`sssControlEmergency` is a special control application that is very similar to `sssControlProduction`. However, it specifies the load libraries needed for emergency fixes rather than the normal production load libraries.

Single control application

A second option is to collect all the control information into a single application. This can make it easier to maintain the control information. The parts might be named as follows:

```
sssCommonMvsCicsGenOpts  
sssXxxMvsCicsGenOpts  
  
sssCommonVseCicsGenOpts  
sssXxxVseCicsGenOpts  
  
sssXxxMvsResourceAssociations  
  
sssXxxVseResourceAssociations  
  
sssXxxMvsLinkageTable  
sssXxxVseLinkageTable  
  
pppppp.BDC    (for bind information that is unique to a program)  
pppppp.LED    (for linkage editor information that is unique to a program)
```

In this naming scheme:

sss Is the subsystem name.

Common

Is used for control information that is common to test and production.

Xxx Is the level of test or production (for example, Dev, SysT, AccT, Prod, or Emergency).

PPPPPPP

Is the name of a program.

Generation options

/DESTNAME, /COBOL, and /NLS are gone.

/BIND and /LINKEDIT have changed. They do not point to a directory, just to a suffix. For example, if your naming convention for linkage editor control parts is ppppppp.LED, where ppppppp is the program name, you should set /LINKEDIT=LED for the generation option.

/OPTIONS, /LINKAGE, and /RESOURCE have also changed. They do not point to a directory, just to a part. For example, you might set /OPTIONS=sssMvsCicsGenOpts.

Using configuration maps

Configuration maps are a way of defining a group of application editions that should be loaded as a group into your image. For example, you might have a configuration map that represents the application versions that are currently in your production environment. When defining configuration maps, consider the following:

- Try to avoid having the same application appear in multiple configuration maps.
- Establish a naming convention for your configuration maps. For example, if you decide to use different configuration maps for development, system test, acceptance test, and production, you might use the following as the configuration map names:

```
Configuration Map Name:  xxxxxxxxDevTestCfmap  
                        xxxxxxxxSysTestCfmap  
                        xxxxxxxxAccTestCfmap  
                        xxxxxxxxProdCfmap
```

```
Version Names:          could just be sequential
```

In this naming scheme, xxxxxxxx is the subsystem name.

If you decide to use a single configuration map and use a build approach in which you build (generate programs and package views) once a week, you might use the following:

Configuration Map Name: xxxxxxxxCfmap

Version Names rrr nn.p

In this naming scheme:

- xxxxxxx Is the subsystem name.
- rrr Is the release name.
- nn Is the build within the release.
- p Is a point between builds.

Use the same version naming scheme for the applications and classes. The p allows the class owners and application managers to create as many versions as they need to between builds. The rrr nn means that the classes, applications, and configuration maps changed during the same week all carry the same release and version number. This helps everyone remember when changes were made.

Migrating GUIs

VisualAge Generator 4.0 provides more than one way to migrate existing GUI parts to the Smalltalk platform. See the following chapters for step-by-step instructions on how to migrate your GUI parts.

- **MSL Migration Assistance Tool:** “Chapter 26. The MSL Migration Assistance Tool on Smalltalk” on page 241 and “Chapter 29. Running the MSL Migration Assistance Tool on Smalltalk” on page 293
- **VAGen Import:** “Chapter 30. Using VAGen Import to migrate VAGen 2.x and Cross System Product code to Smalltalk” on page 329

Regardless of whether you use the MSL Migration Assistance Tool or VAGen Import, the following considerations apply when migrating GUIs to views:

- Each GUI becomes a class in VisualAge Smalltalk. Therefore, the GUIs are not visible in the **VAGen Parts** pane or in the **VAGen Parts Browser**.
- Because VisualAge Smalltalk does not permit national characters such as \$, #, or @ in class names, you must rename GUIs that contain these characters in their names before you attempt to migrate them to VisualAge Generator 4.0. If you do not rename them, a walkback will occur during migration.
- GUIs are called **views** or **visual parts** in VisualAge Smalltalk.
- If you have GUIs that contain any multimedia parts (from the Multimedia category) or DDE parts (from the External Functions category), you must load the following features before trying to migrate:

- VisualAge: Multimedia
- VisualAge: DDE Support
- You might receive a warning message from the MSL Migration Assistance Tool or from **VAGen Import** that says:
 - name failed to load reason

where:

name Is the name of your GUI (view)

reason Is optional

If a reason is given, it indicates that either the Multimedia or the DDE Support features are required for this view. These features must be loaded before you will be able to migrate this GUI.

If no reason is given, you might be trying to migrate a GUI that contains an OS/2 part, but you are running on a Windows NT system. You must modify the GUI so that it no longer contains the OS/2 part before you can migrate the GUI to Windows NT. Alternatively, you can migrate the GUI on OS/2, but you will not be able to use it from a Windows NT client.

- If you have GUIs that contain any parts from the OS/2 category, you should migrate them using the OS/2 version of VisualAge Generator 4.0. This results in fewer migration problems because the OS/2 applications needed to support the OS/2 category parts are only available on the OS/2 platform.

After migration:

- To change the view on OS/2, you must first load the configuration map called *VAGen VA Edit Obsolete*.
- Before trying to load the application containing the view on Windows NT, you must change the view to be platform independent.
 - If the old view used an OS/2-Windows Notebook, you can change it to a portable notebook by selecting **Morph into** from the context menu for the part.
 - If the old view used an OS/2 Container, you can replace it with one of the Container parts within the Lists category.
- Both the MSL Migration Assistance Tool and **VAGen Import** write a list of important changes to a file named *hptguicv.log*. This log file is located in one of the following:
 - The directory specified by the **TEMP** environment variable
 - If **TEMP** is not specified, the directory specified by the **TMP** environment variable
 - If neither **TEMP** nor **TMP** is specified, the current working directory

The warning messages in this log indicate changes that you must make to the view after migration. See “Tasks you must do after migrating GUIs” on page 234 for examples of the changes required.

- After GUIs are imported into VisualAge Smalltalk using either VAGen Import or the MSL Migration Assistance Tool, they cannot be exported as external source format files. Therefore, the views cannot be used in VisualAge Generator 2.2 or earlier. This means that if you migrate a subsystem that shares embedded GUIs with a subsystem that you will migrate at a later time, you have the following alternatives for maintenance of the embedded GUIs:
 1. Do all maintenance on VisualAge Generator 2.2 or earlier using MSLs and when you are satisfied with the change:
 - a. Export an external source format file from the MSL for the changed parts.
 - b. Create an edition of the ENVY applications to which the changed parts belong.
 - c. Import the external source format file into ENVY using the **Defined application** radio button so the changes will go into the same ENVY application in which the parts are already located.
 2. Make the same changes to both the view in ENVY using VisualAge Generator 4.0 and to the corresponding GUI in an MSL using VisualAge Generator 2.2. Due to the changes that occur when GUIs are migrated to views, you might not be able to make identical changes to the view and its corresponding GUI. For example, in VisualAge Generator 4.0, the visual table part no longer appears on the parts palette. Additional changes to the GUIs are described below.

VAGen Container Details and VAGen Variable have equivalent VisualAge Smalltalk parts. Refer to the *VisualAge Generator Programmer's Reference* for details about the differences between the VisualAge Generator and VisualAge Smalltalk versions of these parts.

- The visual table part from the Data Entry category on the parts palette is now obsolete. The visual table part will continue to work in the migrated GUIs, but no longer appears on the parts palette. For new views, use the Container Details part, which provides function similar to the visual table part.

Conversion of GUIs to VisualAge Generator 4.0

Conversion is the process of transforming an existing GUI to a new view in VisualAge Generator 4.0. VisualAge Generator 2.2 and earlier added some features to VisualAge Smalltalk parts, some of which are now in the base VisualAge Smalltalk and some of which are part of the VisualAge Generator extension to VisualAge Smalltalk. The features that are part of the VisualAge Generator extension are prefixed with *VAGen* and therefore, they must be renamed during conversion. Many other tasks are also performed during

conversion to maximize the chance of existing applications working the same as they did in previous releases. The following sections give a complete list of tasks performed during conversion.

Note: The tasks performed during conversion occur automatically when you commit parts to ENVY using the MSL Migration Assistance Tool or when you import parts into ENVY using **VAGen Import**.

Tasks that take place during GUI conversion

The following occur during GUI conversion:

1. Each GUI is created as a subclass of *HptAppBldrView*, which is the VisualAge Generator subclass of *AbtAppBldrView*. *HptAppBldrView* is needed to provide the following features that were added to GUIs in VisualAge Generator 2.2 or earlier releases:

```
#message:title:iconType:buttonType:  
#message:title:iconType:buttonType:helpFile:helpTopic:
```

Because there are other ways to accomplish these actions now, these features are obsolete in VisualAge Generator 4.0 and are no longer available at edit time.

2. Parts are renamed. Table 17 shows the relationship between the old and new part names.

Table 17. Part Name Changes after Migrating a GUI to a View

Old Part Name	New Part Name
Record Member	VAGen Record
Table Member	VAGen Table
Process Member	VAGen Function
Statement Group Member	VAGen Function
Callable Function	VAGen Callable Function
Container Details	VAGen Container Details
Variable	VAGen Variable
File Accessor	VAGen File Accessor

3. Features are renamed. Table 18 on page 230 lists all features in different part classes that are renamed in VisualAge Generator 4.0. VisualAge Generator 4.0 extensions to VisualAge Smalltalk parts are prefixed with *VAGen*. For example, *#destroyPart* is renamed to *#destroyView*. This indicates that it is in the VisualAge Smalltalk base, not a VisualAge Generator extension to VisualAge Smalltalk.

Table 18. Feature Name Changes after Migrating a GUI to a View

Old Feature Name (VisualAge Generator 2.2)	New Feature Name (VisualAge Generator 4.0)
Class: AbtAppBldrPart (views and nonvisual parts)	
#inheritsCommunicationSession:	#'VAGen inheritsCommSession:'
#performRequest:	#'VAGen performRequest:' (should use object scripting instead)
#removeTopLevelSubpartNamed:	#'VAGen destroyTopLevelSubpartNamed:'
#topLevelSubpartNamed:	#'VAGen topLevelSubpartNamed:'
#topLevelSubpartNamed:put:	#'VAGen topLevelSubpartNamed:put:'
Class: AbtAppBldrViewWrapper (View Wrapper)	
#backgroundCreatePart	#backgroundCreateView
#createPart	#createView
#destroyPart	#destroyView
#destroyedPart	#destroyedView
Class: AbtBasicView (all visual parts in the Buttons, Data Entry, Lists, Menus, Canvas, and OS/2 categories)	
#height	#'VAGen setHeight'
#width	#'VAGen setWidth'
#x	#'VAGen setX'
#y	#'VAGen setY'
Classes: AbtCompositeView (visual parts that can contain other parts) and AbtContainerDetailsView (Container Details Tree View, Container Details View, Packeting Container Details View, and VAGen Container Details View)	
#removeSubpartNamed:	#'VAGen destroySubpartNamed:'
#subpartNamed:	#'VAGen subpartNamed:'
#subpartNamed:put:	#'VAGen subpartNamed:put:'
#subpartNamed:put:beforePartNamed:	#'VAGen subpartNamed:put:beforePartNamed:'
#subpartNamed:putOpened:	#'VAGen subpartNamed:putOpened:'
#subpartNamed:putOpened:beforePartNamed:	#'VAGen subpartNamed:putOpened:beforePartNamed:'
Class: AbtFormView (Form, Group Box, Window)	
#topLevelEnabled	#'VAGen topLevelEnabled'
Class: AbtShellView (Window)	
#cancelCloseRequest:	#'VAGen cancelCloseRequest:'
#getFocusPart	#'VAGen getFocusPart'

Table 18. Feature Name Changes after Migrating a GUI to a View (continued)

Old Feature Name (VisualAge Generator 2.2)	New Feature Name (VisualAge Generator 4.0)
#getOSFrameHandle	#'VAGen getOSFrameHandle' (now obsolete - do not use)
Class: AbtTextView (Text View)	
#removeSelectedText	#remove (it was renamed in VisualAge Generator 2.2, but this conversion was not performed in that version)
Class: HptLogicPart (VAGen 4GL Parts)	
#'has executed'	#hasExecuted

The following actions are taken to handle the renamed features:

- Connections referencing old feature names are changed to reference new names.
- Promoted features of old feature names are changed to reference new names.
- All settings applied to old feature names are changed to apply to new names.

Important Notes:

- a. Renaming features connected through a variable — For a variable or tear-off part, because it is unknown at migration time what type of parts it will hold at runtime, VisualAge Generator 4.0 applies all feature name changes listed above to a variable and tear-off part. For example, suppose you have the connection:

```
Button1(#clicked) -> Variable1(#x)
```

is converted to:

```
Button1(#clicked) -> Variable1('#VAGen setX')
```

If #x is a user-defined feature (promoted feature) and not the action #x of *AbtBasicView*, this conversion is wrong, and must be corrected before the application works correctly again. However, the chance of collision is probably very small. When converting a GUI, all renaming events that involve a variable are logged to the *hptguicv.log* file.

- b. #performRequest — Although VisualAge Generator tolerates old feature names used with #performRequest so that current applications will continue to work, functions that set up data items used with this action need to be changed to use new feature names.

4. Attribute-to-attribute connections to a View Wrapper (not connections inside the GUI wrapped by the View Wrapper) where the View Wrapper is the source are reversed to maintain VisualAge Generator 2.2 behavior: when the view being wrapped is created, the connections to the wrapper are aligned as if the wrapper is the target regardless of how the connection was made. In VisualAge Smalltalk 5.0 (VisualAge Generator 4.0), View Wrapper's two-way attribute-to-attribute connections are initialized consistently with all other two-way attribute-to-attribute connections: if the source is not nil and the target is not read-only, the target is aligned with the source's value.
5. An attribute connection to a parameter of another connection is converted to a parameter-to-attribute connection, which is one of the connection types newly created to specifically handle parameters. In VisualAge Smalltalk 5.0, any connection made to a parameter is created as a parameter connection. This conversion ensures that the correct data type conversion for the VAGen data part can take place.
6. In VisualAge Generator 2.2 and previous releases, the default of *inheritsCommSession* was false. VisualAge Generator 4.0 sets the default to true for any new views you define because this is the recommended technique. However, to maintain the same behavior for existing GUIs that are being converted to VisualAge Generator 4.0, if *inheritsCommSession* is not set, conversion sets it to false. Refer to the *VisualAge Generator Programmer's Reference* for more details on *inheritsCommSession*.

Obsolete parts maintained for compatibility

The following parts have been maintained for compatibility, but no longer appear on the parts palette:

- VAGen View Wrapper
- VAGen Object Factory

View wrapper: The **View Wrapper** available in VisualAge Generator 2.2 or previous releases was the VisualAge Generator version, not the VisualAge Smalltalk version. The difference between the two is in the signalling of events in connection initialization. During the process of initializing its parent part's attribute-to-attribute connections that involve it, the VisualAge Generator View Wrapper does not signal events, but instead holds on to them and signals them in order after the initialization is done.

To see the difference between the VisualAge Generator version and the VisualAge Smalltalk version, examine the following example:

```
ParentView(string1) <-> ViewWrapper(string1)

ViewWrapper(string1) -> SETSTRING2_STMTGRP(execute)
                        (which is in ParentView and does nothing)

SETSTRING2_STMTGRP(hasExecuted) -> ParentView(string2)
                        (The parameter of this connection)
```

is connected to the ViewWrapper's string2)

```
ParentView(string1)<->ViewWrapper(string2)
```

If at the time the View Wrapper initializes the connection, you have:

```
ParentView(string1): 'A'  
ParentView(string2): 'B'  
ViewWrapper(string1): nil  
ViewWrapper(string2): nil
```

The results are:

	VisualAge Smalltalk View Wrapper	VisualAge Generator View Wrapper
ParentView(string1):	'A'	'A'
ParentView(string2):	nil	'A'
ViewWrapper(string1):	'A'	'A'
ViewWrapper(string2):	'A'	'A'

However, the VisualAge Generator implementation, which can potentially reduce the number of events signalled, also causes each attribute-to-attribute connection to the View Wrapper to align twice. This side effect is explained in the following example:

```
ParentView(attribute1) <-> ViewWrapper(attribute2)
```

When the View Wrapper creates the view, it causes the connection to be initialized.

With the VisualAge Smalltalk View Wrapper:

- ViewWrapper's attribute2 gets the value of ParentView's attribute1
- ViewWrapper signals that attribute2 has changed. Because the connection is in the middle of being aligned, the signalling of attribute2 does not cause ParentView's attribute1 to be aligned with ViewWrapper's attribute2, and therefore, ParentView's attribute1 does not signal its changed event.

With the VisualAge Generator View Wrapper:

- ViewWrapper's attribute2 gets the value of ParentView's attribute1.
- The event of attribute2 changed is pended until all other connections get initialized.
- After all connections are initialized, ViewWrapper now signals its attribute2 has changed. This causes ParentView's attribute1 to be aligned with ViewWrapper's attribute2 (to the same value).
- ParentView signals that attribute2 has changed, causing other connections to this attribute to execute or align.

Because of the side effect, the VisualAge Generator View Wrapper is maintained in VisualAge Generator 4.0 only to leave existing GUIs working unchanged. Any new View Wrapper dropped is the VisualAge Smalltalk View Wrapper.

Object factory: The **Object Factory** of VisualAge Generator 2.2 or previous releases is now obsolete. After migration, existing GUI applications that use Object Factory still use the old VAGen Object Factory (the one from the palette of VisualAge Generator 2.2, which is *HptObjectFactory*). However, the Object Factory on the palette of VisualAge Smalltalk 5.0 (VisualAge Generator 4.0) is the VisualAge Smalltalk Object Factory (*AbtObjectFactory*). The only difference between the two is that the VisualAge Generator obsolete one, at instance creation time, checks to see if the instance is a GUI, and if it is, it tells the instance to use the communication session owned by its parentPart. For new views, you should use the newly added VisualAge Generator features that have to do with communication sessions to ensure correct behavior.

Tasks you must do after migrating GUIs

You need to do the following after migrating existing GUIs to ensure that they will work:

- Verify that all connections involving variable parts are renamed appropriately. See "Important Notes" on 3a on page 231 for more information.
- Previous releases of VisualAge Generator were more tolerant of connections with mismatched types such as a string connecting to an integer. Because it is a feature of VisualAge Smalltalk, VisualAge Generator 4.0 no longer has control to tolerate these mismatched connections, and therefore walkbacks result. When a GUI is converted to VisualAge Generator 4.0, the type-mismatched connections are logged in *hptguicv.log*. All of the logged connections reflect problems in the views that must be corrected before you test or package your applications.

Example 1: You might have a type-mismatched connection such as:

```
List1(selectionIndex) <----> Button1(enabled)
```

The intention for this connection is to enable *Button1* if there an item selected from *List1*. This connects an integer to a Boolean and will result in a walkback in VisualAge Generator 4.0. You can change this connection to:

```
List1(selectionIsValid) <----> Button1(enabled)
```

Example 2: You might have a mismatched connection such as:

```
List1(selectionIndex) <----> Label1(object)
```

where *Label1* is a string with a value of 1. This connects a string to an integer and results in a walkback in VisualAge Generator 4.0. You can change *Label1* to an integer by doing the following:

- In the VisualAge Organizer window, select **Options** and then **Preferences**. On the **General** page, consider changing the *Preferred Settings View* to **Properties Table (Recommended)**.
- Double-click on *Label1* to open the properties list.
- Select **converter** and then select the ellipsis (...) button that appears beside **converter**.
- Select **Integer** as the *Data type*.
- Press **OK** to close the Converters window.
- Press **OK** to close the Properties window.

VisualAge Generator 4.0 still handles implicit conversion between VisualAge Generator data item types and VisualAge Smalltalk types. Refer to the *VisualAge Generator Programmer's Reference* for more details on the VisualAge Generator Data Part.

- In past releases of VisualAge Generator, fonts were not always sized correctly when a VisualAge Generator dialog was displayed on a Windows NT or Windows 95 system. This problem is corrected in VisualAge Generator 3.1 and later, and all dialog windows now use the Windows NT and Windows 95 standard dialog font. However, if you are unhappy with the fonts on any VisualAge Generator window, you can use the Fonts selection on the Control Panel in Windows NT or Windows 95 to change the window's fonts.

Dual maintenance

The external source format file for 4GL parts that you export from VisualAge Generator 4.0 is not compatible with the external source format file for VisualAge Generator 2.2 or Cross System Product. Therefore, if you migrate a subsystem that shares common parts with a subsystem that you will migrate at a later time, you have the following alternatives for maintenance of the common parts:

1. Maintain the common parts on VisualAge Generator 2.2 or earlier using MSLs and when you are satisfied with the changes:
 - a. Export an external source format file from the MSL for the changed parts.
 - b. Import the external source format file into ENVY using the **Defined application** radio button so the changes will go into the same ENVY application in which the parts are already located.
2. Make the same changes to both the part in ENVY using VisualAge Generator 4.0 and to the corresponding member in an MSL using VisualAge Generator 2.2.

With VisualAge Generator 4.0, you cannot export external source format files for views. See the bullet that begins "After GUIs are imported..." on 228 for information about handling dual maintenance of embedded GUIs.

Migrating from Cross System Product

When you migrate from Cross System Product, you have the following additional considerations:

- If you are migrating from Cross System Product 3.3 or earlier, you are migrating from interpretive CSP/AE to generated COBOL or C++. Refer to *Migrating Cross System Product Applications to VisualAge Generator* (SH23-0244-01) for information on the compatibility considerations involved in this portion of the migration.
- You must also plan for how to handle the workstation environment:
 - Backup and recovery
 - Whether data will reside on the workstation or on the host
 - How to handle calls to non-VisualAge Generator programs when you are using the test facility
 - How to handle functions that were only supported when running the test facility in the CICS environment (for example, CREATX and the use of transient data queues).
- Early releases of Cross System Product allowed invalid data to be stored in the MSLs. If the external source format files contain invalid members, you must either correct the member on Cross System Product and export the external source format file for the member again or correct the external source format file by editing it. The following are examples of some of the types of problems that you might encounter:
 - A data item that has a length of 0
 - A map that contains a NUM field with a date edit, but the field is not long enough to contain a date
 - Members in the MSL that contain generated Cross System Product code that really should have been stored in an ALF.
- VisualAge Generator does not allow generation of applications that check a non-SQL data item for the NULL state. VisualAge Generator rejects as invalid any conditional statement (IF, TEST, WHILE) that tests the state of a non-SQL data item for the NULL condition. However, this type of statement was allowed in Cross System Product 3.3. Therefore, you should rewrite any statements that check non-SQL data items for the NULL state before migrating Cross System Product-generated applications that contain them.

Migrating from OS/2 to Windows NT

If you are changing from the OS/2 to the Windows NT development platform, be sure to review “Changing from OS/2 to Windows NT” on page 298 before you start to migrate.

Using the migration log file

The migration log file is named **mslmig.log** and is located in the same directory as your image. This log file stores the results of **Write to File** operations. You can use a text editor to view and print the log file.

Chapter 25. Pre-migration checklist

1. Before you migrate, you should first read the following:
 - “Chapter 23. Comparing MSLs and ENVY on VAGen 4.0 on Smalltalk” on page 207
 - “Chapter 24. General migration considerations for VAGen 2.x and Cross System Product to Smalltalk” on page 215
 - “Chapter 26. The MSL Migration Assistance Tool on Smalltalk” on page 241
 - “Chapter 27. Migrating production and work-in-progress MSLs to VAGen 4.0 on Smalltalk” on page 255
 - “Chapter 28. VAGen on Smalltalk case studies based on various MSL structures” on page 265
 - VisualAge Generator 4.0 readme file

2. Load any features you need for migration.

If you are migrating GUIs from MSLs to VisualAge Generator 4.0 on Smalltalk, you might need to load the VisualAge Smalltalk Multimedia or DDE Support features before you migrate.

Note: If you use the multimedia feature, refer to the VisualAge Smalltalk installation documentation and readme file for information on how to obtain this feature.

3. Save a clean copy of your VisualAge Smalltalk image. This clean copy should not contain any of your application code. Copy *abt.icx* to *abtclean.icx*, and store the clean copy on your LAN so that all developers have access to it. Saving copies of the *hpt.ini*, *abt.ini*, and *mgr50.dat* files is also recommended.
4. Check with IBM support to see if there are any fixes available for the VisualAge Generator 4.0 MSL Migration Assistance Tool. If there are, import the fixes and then load them into your 4.0 image.
5. For better performance during migration, you might want to do the following:
 - Dedicate a single workstation or server to use for migration. Attempting to use multiple workstations limits VisualAge Generator’s ability to detect duplicates and common code and to determine when previously missing parts have been found. Having the product provide this information more than compensates for any potential time savings that might be gained by using several workstations.

- Copy your MSLs to the workstation you set up to run the VisualAge Generator migration tools, or make sure your code libraries are on a server that your migration workstation can access. This improves performance for the migration tools.
- You will probably use the MSL Migration Assistance Tool to migrate to VisualAge Generator 4.0. Run the tool on the server where the VisualAge Generator 4.0 ENVY library manager is located. This improves performance for committing parts to ENVY.

If you plan to use the MSL Migration Assistance Tool, review “Collecting your source code” on page 296 and “Handling code page changes” on page 297 for information on how to make your code from Cross System Product or previous versions of VisualAge Generator available to the MSL Migration Assistance Tool.

6. Contact your local IBM representative to learn more about VisualAge Generator service offerings and how they can help you with migration.

Chapter 26. The MSL Migration Assistance Tool on Smalltalk

The MSL Migration Assistance Tool is designed to assist in migrating MSLs to the ENVY library manager.

Note: Although the MSL Migration Assistance Tool can help determine when members are not found, it cannot locate missing members. Similarly, the MSL Migration Assistance Tool can help determine when duplicates of a given member exist, but it cannot make the determination as to which is the correct or current level of the member.

You can run the MSL Migration Assistance Tool from either OS/2 or Windows NT.

The migration process works from an MSL directory structure. In some cases, you will not have an MSL on the workstation. For example, if you are migrating from Cross System Product, your MSLs are VSAM files on the host. The MSL Migration Assistance Tool enables you to create MSL directories from external source format files. You can then use these MSL directories during your migration. You do not need to install VisualAge Generator 2.2 to create the MSL directories to use during migration.

The MSL Migration Assistance Tool enables you to select parts (members) from an MSL or MSL concatenation, group them into an application or series of applications, and move them to a "sandbox". When you move a part to the "sandbox", its associates also move. For a GUI, its referenced embedded GUIs and external GUIs and their associates are moved. The applications in the sandbox are in the current image, but are not in the ENVY library manager yet. This allows you to manipulate the applications and to rearrange the VAGen parts within the applications until you are satisfied with the organizational structure. Only one version of a part can be in the sandbox at a time. Each part can only be in one ENVY application within the sandbox.

After you are satisfied with your organizational structure for the applications, you can commit the applications to the ENVY library manager. Committing the applications creates an edition of the applications in the ENVY library manager, creates any needed VAGen part classes for the 4GL member types, creates views for GUIs, and creates the VAGen parts for the 4GL members.

After the applications are in ENVY, you can use any of the ENVY library management functions such as:

- Versioning and releasing the VAGen part classes and parts
- Versioning and releasing the applications
- Creating configuration maps
- Dividing applications into subapplications
- Assigning a class owner or an application manager

Overview of using the MSL Migration Assistance Tool

You can start the MSL Migration Assistance Tool from the VisualAge Organizer window by selecting **Tools** and then **VAGen MSL Migration**. The MSL Migration Part List window is displayed, as shown in Figure 24.

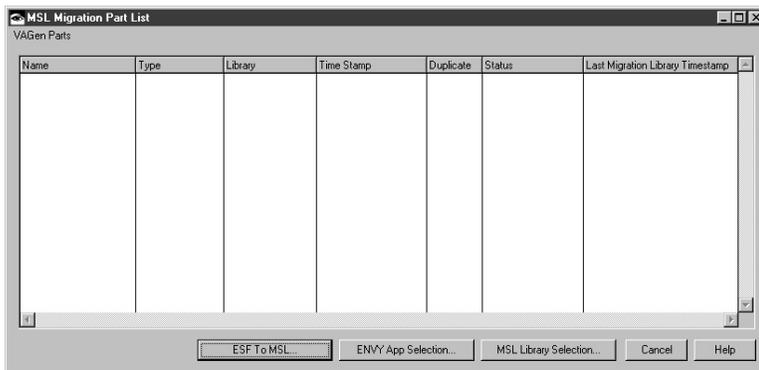


Figure 24. MSL Migration Part List window

The sections that follow provide an overview of how to use the MSL Migration Assistance Tool and the purpose of each window. Reading these sections will help you understand the following chapters that describe how you might organize your members into ENVY applications:

- “Chapter 24. General migration considerations for VAGen 2.x and Cross System Product to Smalltalk” on page 215
- “Chapter 27. Migrating production and work-in-progress MSLs to VAGen 4.0 on Smalltalk” on page 255
- “Chapter 28. VAGen on Smalltalk case studies based on various MSL structures” on page 265

After you have read the chapters on organizing your members into ENVY applications, you will be ready to use the more detailed procedures provided in “Chapter 29. Running the MSL Migration Assistance Tool on Smalltalk” on page 293.

Building MSL directories

Before you can begin the migration process, you must have MSLs to migrate. There are some situations in which you will not have MSLs on the workstation or you cannot use the MSLs for migration. This occurs in the following situations:

- You are migrating from Cross System Product, so you have never had MSLs on the workstation. Create one external source format file for each host MSL and download the external source format files to the workstation.
- You are migrating from VisualAge Generator but have used TeamConnection and do not have MSLs. You might want to create one external source format file for each component to help preserve the organizational structure you created in TeamConnection.
- You are migrating from VisualAge Generator but are changing from an OS/2 development environment to a Windows NT development environment. You cannot use the OS/2 MSLs due to code page differences between the two environments. Create one external source format file for each MSL. See “Changing from OS/2 to Windows NT” on page 298 for information on converting between code pages using these external source format files before you create MSLs on Windows NT.

After you have created your external source format files, from the MSL Migration Part List window, select the **ESF to MSL** push button. The MSL Migration Assistance Tool prompts you for the name of an external source format file, as shown in Figure 25.

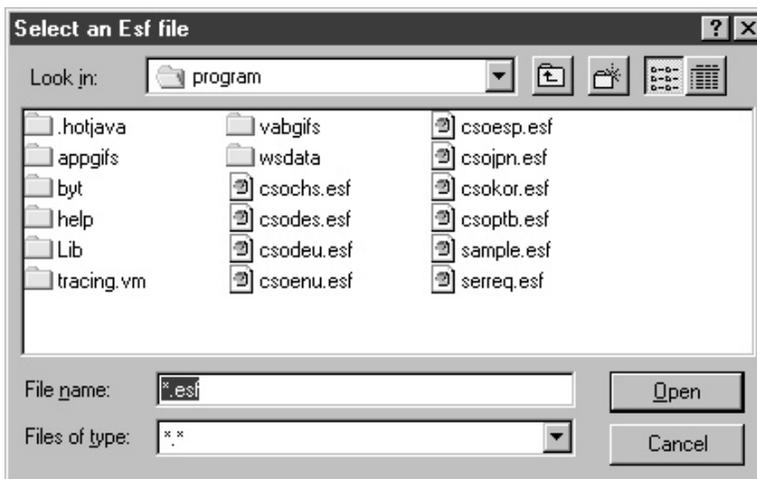


Figure 25. Prompting for an external source format file name

The MSL Migration Assistance Tool then prompts you for the name of a directory into which the MSL members are to be placed, as shown in Figure 26.

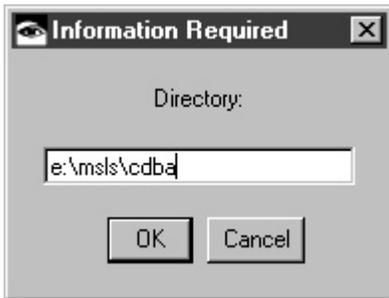


Figure 26. Prompting for an MSL directory name

After you have specified this information, the MSL Migration Assistance Tool creates the MSL members from the external source format file. If you specify a directory that does not exist, the MSL Migration Assistance Tool creates it for you.

Note: The VisualAge Generator 4.0 automatic conversions occur at this time. Therefore, processes and statement groups will become functions.

Selecting MSLs - using the MSL Library Selection window

Before you can begin moving parts to the sandbox, you first need to specify the MSLs that you want to process. From the MSL Migration Part List window, select the **MSL Library Selection** push button. The MSL Library Selection window is displayed, as shown in Figure 27 on page 245.

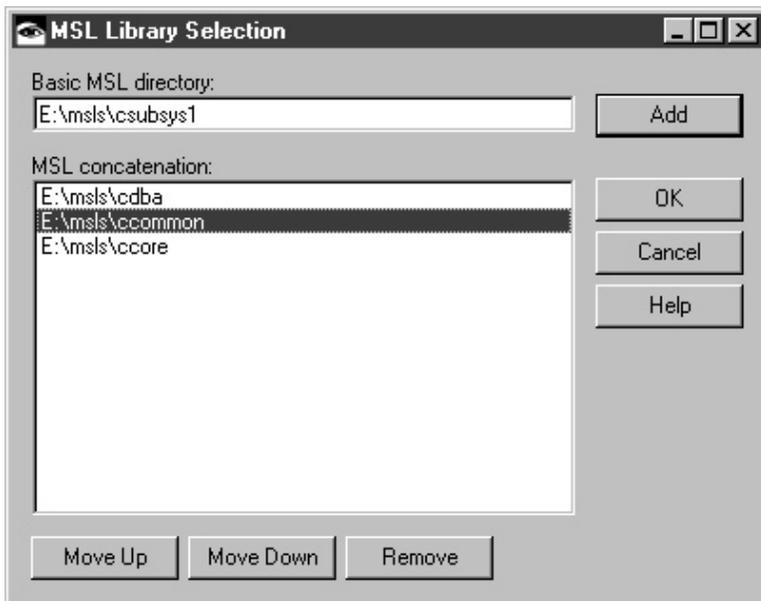


Figure 27. MSL Library Selection window — Adding an MSL to the concatenation

From the MSL Library Selection window, you can specify the drive and path for one or more MSL directories. Specify the drive and path for one MSL in the *Basic MSL directory* field and then select the **Add** push button. Repeat this until you have specified the information for all the basic MSLs in the concatenation sequence that you want to process. You can use the **Move Up**, **Move Down**, and **Remove** push buttons to change the concatenation sequence.

After you specify the MSLs and the MSL concatenation sequence, select the **OK** push button. The Part List Selection Criteria View window is displayed and lists the MSL directories.

Selecting parts for a part list - using the Part List Selection Criteria View window

The Part List Selection Criteria View window is similar to a Member Selection List in Cross System Product or VisualAge Generator.

The Part List Selection Criteria View window is shown in Figure 28 on page 246.

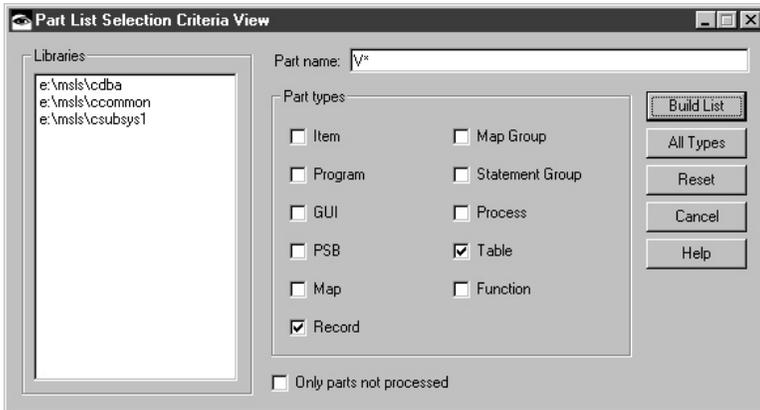


Figure 28. Part List Selection Criteria View window

From the Part List Selection Criteria View window, you can do the following:

- Specify a portion of a part name using a wildcard.
- Specify which part types you want to appear on the MSL Migration Part List window.

However, the Part List Selection Criteria View window differs from a Member Selection List in the following ways:

- It allows you to specify whether you want to see all parts that satisfy the part type criteria or whether you only want to see those parts that satisfy the part type criteria that have not yet been moved to the sandbox. This is controlled by the **Only parts not processed** toggle button.
- It does not allow you to specify which MSLs from the concatenation sequence are to be used. All listed MSLs are always used. If duplicate members exist, they are all shown.
- **Function** is listed as a part type. If you are migrating using external source format files to create pseudo MSLs, select the function part type. If you are migrating using your VisualAge Generator 2.x MSLs, select the **Process** and **Statement Group** part types.

When you select the **Build List** push button, the parts that satisfy the selection criteria appear in the MSL Migration Part List window.

Selecting parts to move to ENVY - using the MSL Migration Part List window

The MSL Migration Part List window lists the parts that satisfy the selection criteria specified in the Part List Selection Criteria View window.

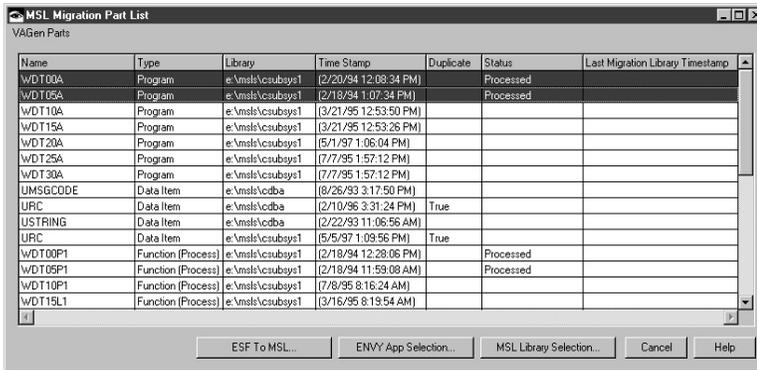


Figure 29. MSL Migration Part List window with parts

From the MSL Migration Part List window, you can select the parts that are to be moved to the sandbox. The MSL Migration Assistance Tool moves the selected parts **with their associates** to the sandbox. You can specify that the selected parts are to be moved in the following ways:

- Into a single application
- Into multiple applications, with each selected program becoming a different application
- Into an application that is already in the sandbox

When you move parts to the sandbox, it is possible that an associate of the part you are currently moving is already in the sandbox, but in a different application from the one that you have specified as the target for this move.

In this case, the associates that are shared by more than one application are automatically moved into a separate application called ApplicationNoden, where *n* is a number to distinguish different application nodes. Each ApplicationNode represents a group of parts that have the same set of required applications.

When you move one or more parts to the sandbox, the VG Part Prerequisites View window is displayed and shows the current state of the sandbox.

For the purposes of migration, the associates of a part are the same as in Cross System Product or VisualAge Generator 2.x. However, to assist in setting the prerequisites for views (GUIs) correctly, the MSL Migration Assistance Tool also moves external GUIs and their associates when you move a GUI to the sandbox.

Note: Process parts and statement group parts are displayed in the MSL Migration Part List window as *function* parts. If you used the **ESF to MSL** push button to create an MSL that you could migrate, all your

process and statement group parts were converted to function parts when the MSL was created. These parts are displayed in the MSL Migration Part List window with a type of **function**. The MSL Migration Assistance Tool is unable to discern the origin of these function parts. If, however, you used the **MSL Library Selection** push button to select an existing MSL for migration, your process and statement group parts are displayed in the MSL Migration Part List window with a type of **function (process)** or **function (statement group)**. This allows you to determine which function parts were originally process parts and which were originally statement group parts.

After you think you have moved all the parts to the sandbox, you can verify this from the Part List Selection Criteria View window by selecting the **Only parts not processed** toggle button, then selecting the **All Types** push button, and then selecting the **Build List** push button. If everything has been moved to the sandbox, the MSL Migration Part List window will not list any parts.

Special columns in the MSL Migration Part List window

The MSL Migration Part List window provides information to help in determining which parts to move to the sandbox. The **Status**, **Duplicate**, and **Last Migration Library Timestamp** work together to help you resolve duplicate or missing parts.

The **Status** column indicates the following:

Status	Meaning
<blank>	A part with the same name and timestamp has not yet been moved to the sandbox. The Duplicate column indicates whether a part with the same name but a different timestamp has been moved to the sandbox.
Processed	A part with the same name and timestamp has been moved to the sandbox but has not yet been committed to ENVY.
Migrated	A part with the same name and timestamp has been moved to the sandbox and committed to ENVY. The sandbox has not been reset from ENVY as described in “Resetting the MSL Migration Assistance Tool sandbox” on page 252.
Not Found	The part is an associate of a part that was already moved to the sandbox from a different MSL concatenation sequence. This part was not found in that previous concatenation

sequence, and thus is identified in the sandbox as a *Not Found* part. You can update the sandbox with this newly found part.

In ENVY Only

The part is in your ENVY image, but has not been loaded into the sandbox from either an MSL or from ENVY. This might occur if you have deleted all the applications from the sandbox and have not yet reset the sandbox from your ENVY image.

The **Duplicate** column indicates *True* if the part is a duplicate:

- Another part with the same name but a different timestamp is already in the sandbox.
- Another part with the same name is in the same MSL concatenation sequence.

When **Duplicate** is *True*, the **Last Migration Library Timestamp** provides the timestamp of the part in the sandbox to help in determining which of the duplicates you really want to migrate to ENVY.

The value displayed in the **Last Migration Library Timestamp** depends on whether the sandbox has been reloaded from ENVY:

- If the sandbox has not been reloaded (for example, you are working with your initial set of MSLs or continuing with additional MSLs after committing some applications), then the timestamp is the timestamp from the MSL of the part in the sandbox.
- If you have reloaded the sandbox from ENVY (for example, you migrated one subsystem several months ago and now are ready to do the remaining subsystems), then the timestamp is the timestamp of the edition of the part that is currently loaded in your image.

The **Last Migration Library Timestamp** is not displayed in the following situations:

- A version of the part is not in the sandbox. This occurs if the duplicates are both in the current MSL concatenation sequence or when you have cleaned out the sandbox, but have not reloaded from your ENVY image.
- The part in the sandbox is identified as a *Not Found* part. There is no timestamp available for a *Not Found* part.
- The part in the MSL Migration Part List window has the same timestamp as the part in the sandbox.
- The part is not a duplicate.

Other MSL Migration Part List functions

From the MSL Migration Part List window, you can perform the following tasks:

- Add a previously *Not Found* part that you have now found to the sandbox.
- Resolve duplicates by:
 - Removing the duplicate from further consideration so that it no longer appears on the MSL Migration Part List window and will not be considered when looking for associates of other parts.
 - Replacing a part in the sandbox with a different version of the part.
- Add new parts or replace existing parts in applications that are in the sandbox, but which have already been committed to ENVY. This enables you to update an existing ENVY application with a new version of the part.

Working in the sandbox - using the VG Part Prerequisites View window

The VG Part Prerequisites View window shows the ENVY applications that have been created in the sandbox. The number of parts in the application is shown to the right of the application name. When you select one of the applications from the left pane, the parts contained in the application and the required and dependent applications for the application appear in the other panes.

Figure 30 shows the VG Part Prerequisites View window.

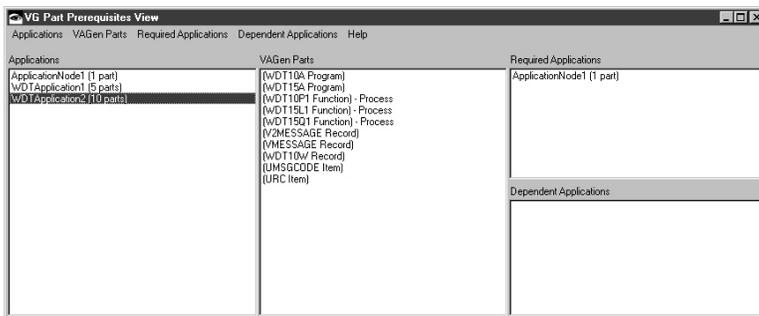


Figure 30. VG Part Prerequisites View window

Identifying common code

The MSL Migration Assistance Tool helps you identify parts that are associated with more than one program. An ApplicationNode is created automatically whenever a common part is used by more than one application. The MSL Migration Assistance Tool does this by automatically moving the common part into a new ApplicationNode. This removes the common part from the original application. Both the original application and the application currently being created specify the ApplicationNode for the common part as a prerequisite in the **Required Applications** pane. If several parts are shared by

two applications, they are all automatically moved to the same ApplicationNode. ApplicationNodes are the mechanism that the MSL Migration Assistance Tool uses to prevent putting the same part into two different ENVY applications. ApplicationNodes help you identify common code that might need to be placed in different ENVY applications rather than in the application in which the part was originally placed.

For example, if record ABC is used by Program1, then ABC is initially included in the same application as Program1. Later, if Program2 is placed in a different application in the sandbox and Program2 also uses record ABC, then record ABC will be automatically moved from the application that contains Program1 and placed in a new ApplicationNode_n, where **n** is a number to distinguish different application nodes.

Identifying missing parts

The MSL Migration Assistance Tool also helps you identify parts that are missing. For example, if you migrate Program1 and it has record DEF defined in its Tables and Additional Records List, record DEF might not exist anywhere in the MSL concatenation sequence. In this case, the record DEF is temporarily included in the same application as Program1, but is identified with the notation *Not Found* in the **VAGen Parts** pane. You can create an application called MyNotFoundApp and move the missing parts to that application as you migrate parts to the sandbox. Alternatively, the MSL Migration Assistance Tool moves any missing parts that are included in an application being committed to ENVY to a NotFoundApp during the commit process. This is because ENVY does not support empty parts.

You can create a list of the missing parts and write the list to the System Transcript window. From the System Transcript, you can save to a file or print the list.

Other sandbox functions

From the VG Part Prerequisites View window, you can perform the following tasks:

- Create a new application. If you have not previously created a CommonDataApp by moving data items, records, and tables to the sandbox, you might decide to create one to contain shared data items and records that you discover when they are moved to ApplicationNodes as you migrate programs and views.
- Mark an application *unexplodable*. Marking an application *unexplodable* means that if parts in this application are used by a program that is moved to the sandbox later, they are not moved to a new, shared ApplicationNode. For example, if you have a CommonDataApp that contains all your common data items, records, and tables, you should mark it as *unexplodable* so that these common parts are not continually moved to new ApplicationNodes as you migrate other programs.

- Collapse an application into another application. If you decide that two separate applications would be more useful if they were combined into a single application, the collapse function allows you to merge one application into the other.
- Rename an application if you change your mind about the name.
- Check the consistency of an application to ensure that all required applications are specified.
- Normalize an application to ensure that only the applications that are needed are listed as required applications. This avoids unnecessary prerequisites being created in ENVY and is most beneficial when views (GUIs) are being migrated.
- Change the required applications for an application.
- When you are satisfied with the organization of your applications, you can commit them to ENVY. This creates the applications, classes, and VAGen parts in ENVY. When a GUI (view) in one application uses an embedded or external GUI in another application, the prerequisites for the ENVY application are also established.

If you have modified an application that is already in ENVY, the commit process creates a new edition of the application and any affected classes and VAGen parts. New classes and VAGen parts are created within the application as needed.

Resetting the MSL Migration Assistance Tool sandbox

You might want to migrate one subsystem, work with it for a while in ENVY to gain some experience, and then migrate your other subsystems. If you do this, you need to reset the MSL Migration Assistance Tool from ENVY to reflect the parts that are currently in ENVY. For example, if you added any new parts to ENVY, these new part names might also exist in an MSL that you want to load into the sandbox for migration to ENVY. These parts in the MSL would thus need to be treated as duplicate parts.

Use these steps to load applications into the sandbox from ENVY:

1. From the VisualAge Organizer window, load into your image any applications (such as your common applications) that you need to have in the sandbox when you migrate your new MSLs.
Alternatively, from the Configuration Maps Browser, load the configuration maps that contain the applications you need in your image.
2. From the MSL Migration Part List window, select the **ENVY App Selection** push button.

If there are applications already in the sandbox, a message is displayed asking if the applications can be deleted. By deleting the applications currently in the sandbox (from your last migration operation), you can

ensure that you start from a consistent set of applications (all loaded from MSLs or all from ENVY). If you choose not to delete the applications, your new selections are added to the existing sandbox list.

3. After you respond **Yes** or **No** to the message about existing applications, a Selection Required window appears. In this window, select the applications from your image that you want to have in the sandbox when you continue your migration with your other subsystems. For example, in most situations, you will want to load your common applications. Loading common applications avoids the MSL Migration Assistance Tool identifying these common parts as *Not Found* when you migrate your other subsystems.
4. From the MSL Migration Part List window, select **MSL Library Selection** to load into the sandbox the next group of MSLs that you want to migrate.

Notes:

1. When applications are loaded into the sandbox from ENVY, the timestamp for their parts in the sandbox reflects the timestamp of the parts in your ENVY image, not the timestamp of the original MSL member. Therefore, the **Last Migration Library Timestamp** also reflects the timestamp from ENVY.
2. When applications are loaded into the sandbox from ENVY, there is no analysis to determine if these applications use any parts that have not yet been found.
3. You cannot load an application into the sandbox from your ENVY image in the following situations:
 - After migration to ENVY, you added a nonvisual part or a Smalltalk class.
 - After migration to ENVY, you added parts for VisualAge Generator control information (generation options, linkage table, resource associations, bind control commands, or linkage editor control statements).

If you try to load applications that have been changed in these ways, you will receive a message saying:

```
Unsuccessful in reading ENVY app into tool
```

When you receive this message, the System Transcript window lists the parts that prevent the application from being loaded into the sandbox. To load the application into the sandbox, you must first remove the classes that are causing the problem.

If the application and the class are already versioned, use the following steps:

- a. Open a new edition for the application.
- b. Unload the class.

- c. Load the application into the sandbox.
- d. Reload the original application back into your image (so that it is in your image when you version and release classes after committing the next set of applications in the sandbox).

If the application and the class are not versioned yet, use the following steps:

- a. Version the class.
- b. Unload the class.
- c. Load the application into the sandbox.
- d. Reload the class (so that it is in your image when you version and release classes after committing the next set of applications in the sandbox).

Chapter 27. Migrating production and work-in-progress MSLs to VAGen 4.0 on Smalltalk

Your organization might have many MSLs, each used for different purposes. For example, your Database Administrator might be responsible for maintaining one MSL that contains the data item and SQL row record definitions. You might have MSLs that reflect the code that is currently in production, and other MSLs contain the changes that are in the process of being made. These MSLs containing changes include the developers' read/write MSLs, staging MSLs, system test MSLs, and acceptance test MSLs. These MSLs containing changes will be referred to in this document as *work-in-progress* MSLs.

In general, for a system or subsystem, you should migrate your production MSLs first and then migrate your work-in-progress. There are different recommended techniques for migrating production and work-in-progress MSLs.

Migration of your production MSLs involves using the MSL Migration Assistance Tool. The MSL Migration Assistance Tool helps you structure your VisualAge Generator code into ENVY applications by using the sandbox approach explained in "Chapter 26. The MSL Migration Assistance Tool on Smalltalk" on page 241. This is easier than using **VAGen Import** for external source format files and then trying to rearrange the parts into ENVY applications.

Migration of your work-in-progress MSLs varies depending on the number of new members in the MSLs:

- If all (or most) members already exist in the production MSLs, you arranged the parts into applications when you migrated your production MSLs. Therefore, to migrate your work-in-progress MSLs you can use **VAGen Import** to import the external source format files directly into ENVY.

You can use either of the following selection paths to access **VAGen Import**:

- From the **Applications** menu in the VisualAge Organizer window, select **Import/Export** and then **VAGen Import**.
- From **VAGen Parts** menu in the VisualAge Organizer window, select **Import/Export** and then **VAGen Import**.

Note: While other references in this book to **VAGen Import** only describe access through the **VAGen Parts** menu, both paths are usually valid.

VAGen Import allows you to specify that each part in the external source format file is to be placed in the same ENVY application in which it is already defined. This preserves the structure of your ENVY applications that you created using the MSL Migration Assistance Tool.

- If you have many new members that did not exist in the production MSLs, you need to assign these members to applications. For a large number of new members, it is easier to use the MSL Migration Assistance Tool to migrate your work-in-progress MSLs.

The techniques for migrating production and work-in-progress MSLs are explained in:

- “Production MSLs”.
- “Work-in-progress MSLs” on page 261.

You might want to migrate one subsystem, work with it in ENVY for a while and then migrate your remaining subsystems. To do this, you need to reset the MSL Migration Assistance Tool from ENVY so that any changes you have made to previously migrated parts is reflected in the sandbox. This is described in “Resetting the MSL Migration Assistance Tool sandbox” on page 252.

Production MSLs

Migrating production MSLs involves the following general steps:

1. If you are migrating from Cross System Product, do the following:
 - Export external source format files for each production MSL.
 - Download the external source format files to the workstation.
 - Use the MSL Migration Assistance Tool to create an MSL directory structure on the workstation.
2. If you are migrating from previous versions of VisualAge Generator and continuing to use the OS/2 development environment, your MSLs should already exist on the workstation or LAN.
3. If you are migrating from previous versions of VisualAge Generator and changing to use the Windows NT development environment, you must:
 - Export external source format files for each production MSL.
 - Convert the external source format files to the Windows NT code page as described in “Changing from OS/2 to Windows NT” on page 298.
 - Use the MSL Migration Assistance Tool to create an MSL directory structure on the workstation.
4. Move parts into the sandbox as described in “Techniques for moving parts to the sandbox” on page 257.
5. Commit the applications in the sandbox to ENVY.

6. In ENVY, do the following:
 - Version and release the parts.
 - Version the applications.
 - Create configuration maps that reflect the code that is in your production system.
 - Package the views and generate the programs. Then test to be sure that what you migrated matches your production code.

Techniques for moving parts to the sandbox

There are three basic techniques that can be used to move parts into the sandbox. The choice partly depends on how clean your MSLs are in terms of the following:

- There are no missing parts
- All code is currently used
- There are no duplicates

Each of the three techniques involves using the MSL Migration Assistance Tool and migrating only the MSLs that contain the production level code. The three techniques are described in the following sections:

- “All data items, records, and tables are used”
- “All records and tables are used” on page 258
- “Some records and tables might not be used” on page 259

“Considerations for the migration techniques” on page 259 provides information that applies to all three techniques.

All data items, records, and tables are used

If you know the data items, records, and tables are all actually being used, move the parts to the sandbox in the following order:

1. Data items
2. Records and tables
3. Mark the applications created for data items, records, and tables as *unexplodable*, as described in “Other sandbox functions” on page 251.
4. Programs and GUI applications — their associates that have not yet been moved will be moved to the sandbox with them
5. Any processes, statement groups, functions, maps, map groups, and PSBs that are no longer used or that are not used by the current MSL concatenation.

This technique has the advantage of creating the fewest ApplicationNodes when common parts are discovered (see “Identifying common code” on page 250). Because all data items are migrated first, followed by all records

and tables, this technique assumes that all data items, records, and tables are actually used. If this is not the case, there is the disadvantage of migrating parts to ENVY that are never used.

This technique works well if you have an MSL that contains all your common global data items, records, and tables and you know that all these parts are actually used. This might occur if a database administrator maintains the MSL that contains the common global data items, records, and tables.

All records and tables are used

If you know that all records and tables are used, but are not sure whether all the data items are used, move the parts to the sandbox in the following order:

1. Records and tables — their associated data items will be moved to the sandbox with them
2. Mark the application(s) created for the records and tables as *unexplodable*
3. Move any remaining data items to an application called MyUnusedItemsApp. Do not mark this application as *unexplodable* so items that are used directly by applications (for example, called parameters) will be automatically moved to a new ApplicationNode. After the data items are identified, you can then move them to the application containing records and tables. What remains in MyUnusedItemsApp after everything is moved to the sandbox are all the global data items that are no longer used by this MSL concatenation.

If you will be migrating other MSL concatenation sequences that might use these data items, do not commit MyUnusedItemsApp to ENVY until you have migrated the other MSL concatenations. Waiting to commit enables you to move data items that are used by the later concatenation sequences to other applications in the sandbox.

4. Programs and GUI applications — their associates that have not yet been moved will be moved to the sandbox with them
5. Any processes, statement groups, functions, maps, map groups, and PSBs that are no longer used or that are not used by the current MSL concatenation

This technique has the advantage of minimizing the ApplicationNodes that are created automatically. Some data items that are initially thought to be unused might be moved to an ApplicationNode. For example, this occurs when the only use of a global data item is as a called parameter for an application.

Because records and tables are migrated first, this technique assumes that all records and tables are actually used. If this is not the case, there is the disadvantage of migrating records and tables (and their associated data items) that are never used.

Some records and tables might not be used

If you are not sure whether some records and tables are currently used, move the parts to the sandbox in the following order:

1. Programs and GUI applications — their associates will be moved to the sandbox with them
2. Whatever is left — this might involve data items, records, tables, processes, statement groups, functions, maps, map groups, and PSBs that are no longer used or which are not used by the current MSL concatenation

This technique has the advantage of identifying any parts that are no longer used for this MSL concatenation sequence. It has the disadvantage of potentially creating many ApplicationNodes during migration. This is because data items, records, and tables tend to be shared among many applications. You will be identifying common data items, records, and tables when you move applications to the sandbox rather than taking care of them prior to dealing with programs and GUIs as described in the other techniques.

Considerations for the migration techniques

The following notes apply to all three techniques:

- If you have naming conventions that help you identify common parts, you can use your naming conventions to help you separate parts into ENVY applications. For example, if records starting with V* indicate a common record and records starting with W* belong to a program, you might choose to only move common records (the ones starting with V*) when moving records and tables to the sandbox using the first two techniques.
- If you plan to put GUIs and their corresponding server programs into different ENVY applications, consider moving the server programs to the sandbox first. This places any 4GL part that is shared between the GUI and the server program in the same application as the server program. Before you move the GUIs to the sandbox, set the applications that contain the server programs based on how you want to handle the shared 4GL parts. Set the applications as:
 - *explodable* if you want to identify the shared 4GL parts — for example, if you want to put the shared parts into a common code application.
 - *unexplodable* if you want to keep the shared 4GL parts with the server programs.
- When moving a GUI to the sandbox, the MSL Migration Assistance Tool includes its associates and also any external GUIs and their associates. This is to ensure that the prerequisites for the GUI are set according to VisualAge Smalltalk rules. (VisualAge Smalltalk requires an ENVY application that contains a View Wrapper part to list the application that contains the View Wrapper part as a prerequisite. For example, if GUIA uses external GUIB and GUIB is not already in the sandbox, when you migrate GUIA, GUIB will also be put into the sandbox in the same

application as GUIA. If GUIB is already in the sandbox, the application that contains GUIA specifies the application that contains GUIB as a prerequisite.)

If you have a menu system in which the menu GUI includes external GUIs for all your subsystems, do not migrate the menu GUI first. Migrating the menu GUI first would cause your entire system to be moved to the sandbox at the same time. This results in a very large application. Instead do the following:

- Move the GUI for each subsystem to a separate ENVY application.
- Mark the ENVY applications for the subsystems as *unexplodable*.
- Move the menu GUI. The menu GUI will then include the applications created for the subsystems as prerequisites.

If you have several layers of menus, you could start at the lowest level menus or at one of the intermediate levels of menus.

See “Migrating GUIs” on page 226 for more details about what happens when you migrate GUIs to views.

- The order in which you migrate parts to the sandbox affects how parts are assigned to applications. For example, in one small test case, GUIs were moved to the sandbox first and then the server programs. The same test was repeated, but this time moving the programs to the sandbox first. In both tests, the first application was marked as *unexplodable*. The results, in terms of number of parts, were:

Table 19. Number of Parts

Application	Number of Parts - Moving GUIs First	Number of Parts - Moving Programs First
GuiApplication	110	86
ServerProgram	117	141

In both cases, 227 parts were moved to the sandbox. The difference is due to which application contains the parts that are shared by the GUIs and server programs.

- For each MSL or each MSL concatenation sequence, be sure to check that everything in the MSL was moved to the sandbox. Just migrating all the GUIs and programs does not guarantee that everything in the MSL was migrated — there might be records, tables, data items, and so on that are used by applications in other MSL concatenation sequences but are not used in the current concatenation sequence. You can use the toggle button **Only parts not processed** on the Part List Selection Criteria View window to limit the list of VAGen parts to those that have not yet been processed by

the MSL Migration Assistance Tool. For each part that has not been processed, you need to decide whether it needs to be migrated or is no longer used.

- For each MSL concatenation sequence:
 - After committing the applications to ENVY, package the views and generate the programs, map groups, and tables. Then test to be sure that what you migrated matches your production code. For programs, you might want do the following:
 - Generate each program without any of its tables
 - Generate each table

This avoids generating common tables multiple times.

If some programs or tables cannot be generated or if some views cannot be packaged, then this highlights an area where there are problems in migration. To determine whether the problem is in the organization of the ENVY applications or whether the problem is in the original source code, try generating the failing GUI, program, map group, or table on the original Cross System Product or VisualAge Generator system using your original MSLs.

Notes:

1. Any programs for the C++ target environments **must** be regenerated for VisualAge Generator 4.0. There is no coexistence of C++ runtime services between VisualAge Generator 2.2 and 4.0. In addition, it is **strongly recommended** that you regenerate all programs for the COBOL environments and package all views to be sure that you migrated the correct level of code.
2. For CICS OS/2, the default *parmform* option in the linkage table was *COMMDATA*. With VisualAge Generator 4.0, the new option *COMMPTR* is the default. Therefore, if you never specified linkage tables for CICS OS/2, you might need a linkage table now.
 - Test the generated programs.

Work-in-progress MSLs

After you have completed the following tasks, you are ready to migrate your work-in-progress MSLs:

- Migrated your production MSLs using the MSL Migration Assistance Tool
- Versioned and released the parts (views and VAGen part classes)
- Versioned the applications
- Created configuration maps for your production level programs
- Generated the programs and packaged the views
- Tested your migrated production level code

The process for migrating your work-in-progress MSLs assumes you have separate MSLs for staging, system test, and acceptance test as shown in Figure 31.

Depending on the number of new members in your work-in-progress MSLs, use one of the following techniques:

- If there are no (or very few) new members in your work-in-progress MSLs, see “Using VAGen Import” on page 263.
- If there are many new members, see “Using the MSL Migration Assistance Tool” on page 264.

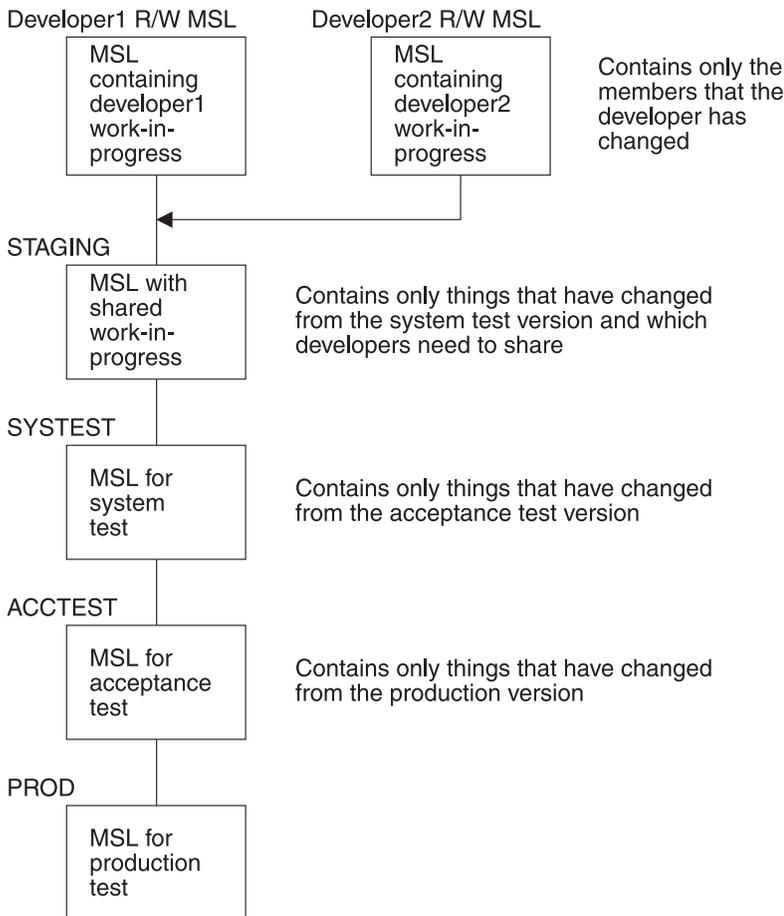


Figure 31. Sample MSL Concatenation for Complete Production MSL and Deltas for Test

Using VAGen Import

If you have no (or very few) new members in your work-in-progress MSLs, you can migrate them using **VAGen Import** by working backward through your MSL concatenation sequence from the production MSLs, and doing the following:

1. Open editions of the applications so that you will be able to import into them.
2. Export external source format files for the acceptance test (ACCTEST) MSL.
3. Use **VAGen Import** to import the external source format files into ENVY, after selecting the **Defined application** radio button. This causes parts that are in the external source format files that already exist in ENVY to be imported into the same application in which they are already defined.
4. Version and release the classes (VAGen part classes and views).
5. Version the ENVY applications. This provides a base line for the code that was in acceptance test.
6. Create a configuration map that contains the version of the applications that are in acceptance test.
7. Package the views and generate the programs, map groups, and tables that have changed for the acceptance test level. Then test to be sure that everything migrated successfully at this level.
8. If there are additional levels of test MSLs, repeat the above steps for each level of testing, working backward through your MSL concatenation sequence. For example, for the MSLs shown in Figure 31 on page 262, you should do the SYSTEST MSL next, followed by the STAGING MSL.
9. After all work-in-progress MSLs have been migrated, other than the developers' read/write MSLs, assign ownership of the configuration maps, applications, and classes.
10. Developers can then migrate their read/write MSLs by doing the following:
 - Open editions of the applications.
 - Export external source format files for their read/write MSL.
 - Use **VAGen Import** to import the external source format files into ENVY after selecting the **Defined application** radio button. This causes parts that are in the external source format files that already exist in ENVY to be imported into the same application in which they are already defined.
 - The developers can then continue with their normal development and test work and wait to version and release the classes until their testing is completed.

Using the MSL Migration Assistance Tool

If you have many new members in your work-in-progress MSLs, it might be easier to use the MSL Migration Assistance Tool so that you have the same flexibility to arrange parts into ENVY applications as you did when you migrated your production MSLs. To migrate your work-in-progress MSLs using the MSL Migration Assistance Tool, you have the following options:

- If you have not made any changes to the parts you have committed to ENVY and your sandbox is still available, you can use the same image and sandbox that you previously used during migration. The advantage of this is that the **Last Migration Library Timestamp** reflects the timestamp from the MSL(s) that you previously migrated.
- If you have made any changes to the organization of your parts within ENVY or you have added or deleted parts from ENVY, then you must reload the sandbox as described in “Resetting the MSL Migration Assistance Tool sandbox” on page 252.

All the functions of the MSL Migration Assistance Tool are available to you to assist in migrating your work-in-progress.

Chapter 28. VAGen on Smalltalk case studies based on various MSL structures

There are several common ways that you might have organized your MSLs. These include the following techniques:

- Techniques for Production MSLs
 - “Multiple subsystems with no duplicates” on page 267
 - “Multiple subsystems with controlled duplicates” on page 269
 - “Separate production MSLs for each developer” on page 272
 - “MSLs that contain unintended duplicates” on page 277
 - “MSLs containing code from VisualAge Generator Templates or BW*Wizard” on page 278
- Techniques for Work-in-Progress MSLs
 - “Complete set of MSLs for production and deltas for test” on page 282
 - “Complete sets of MSLs for test and production” on page 285
 - “MSLs from marketing or other demonstrations” on page 289

Combinations of the above techniques might be used in your organization. In particular, one of the techniques for production MSLs might be combined with one of the techniques for work-in-progress MSLs.

In addition, because Cross System Product and VisualAge Generator had no formal library management, it is common to find unintended duplicates or even triplicates of members in MSLs. An example of this is described in “MSLs that contain unintended duplicates” on page 277.

Understanding the diagrams and terminology

This chapter uses diagrams to represent MSL concatenation sequences. These diagrams are explained in “MSL structure diagrams”.

This chapter also uses the **advance** command in explaining how some of the MSL concatenation sequences were used. The advance command, which was not supported in Cross System Product, is explained in “Advance command in VisualAge Generator” on page 266.

MSL structure diagrams

Figure 32 on page 266 shows a diagram of an MSL structure similar to the ones that are used in the rest of this chapter.

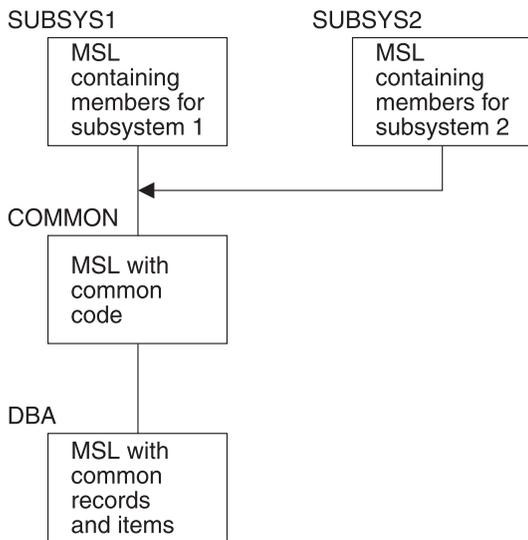


Figure 32. Sample MSL Concatenation

In Figure 32, each box represents a different MSL. The four MSLs are SUBSYS1, SUBSYS2, COMMON, and DBA. The lines between the boxes represent MSL concatenation sequences. Figure 32 shows two MSL concatenation sequences. The first concatenation sequence is SUBSYS1, followed by COMMON and then DBA. The second concatenation sequence is SUBSYS2, followed by COMMON and then DBA.

Advance command in VisualAge Generator

Releases of VisualAge Generator prior to 3.0 provided an **advance** command that was not available in Cross System Product. Advance moves members from one MSL of a concatenation (the source) to the next MSL of the concatenation sequence (the target). After a member is moved to the target MSL, it is deleted from the source MSL. In Figure 32, you can advance a member from SUBSYS1 to COMMON, from SUBSYS2 to COMMON, and from COMMON to DBA. However, you cannot advance a member from SUBSYS1 to DBA.

For Cross System Product, the equivalent function is done by the following steps:

1. Concatenate the target MSL first (read/write) and the source MSL second (read-only).
2. Copy the members from the source MSL to the target MSL.
3. Change the MSL concatenation sequence so the source MSL is first (read/write).
4. Delete the members from the source MSL.

In the sections that follow, advance is used to describe how members are moved from one MSL to the next.

Multiple subsystems with no duplicates

The scenario for separate MSLs for each subsystem has the following MSL structure:

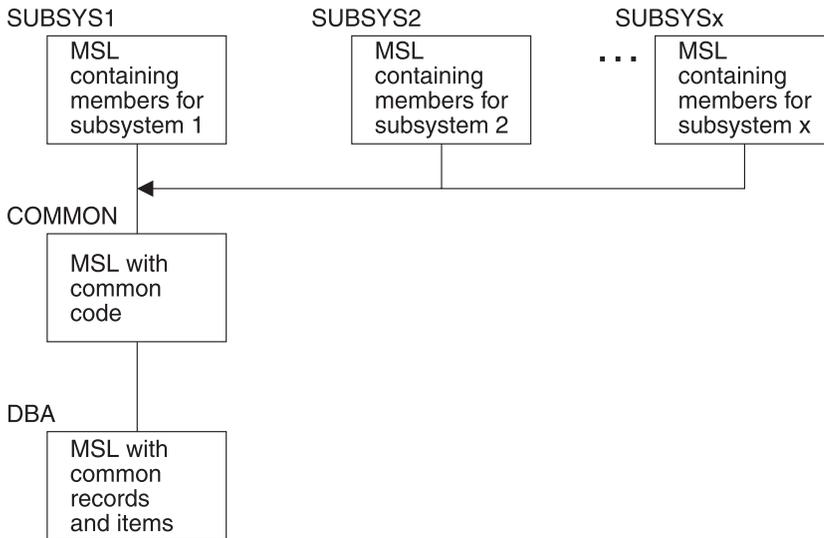


Figure 33. Sample MSL Concatenation for Multiple Subsystems with no Duplicates

In this scenario, there are no duplicate members. Each member exists in one and only one MSL.

Recommendations

Consider migrating the MSLs to ENVY as follows:

1. Verify that there are no duplicates in the MSLs. If there are duplicates, either resolve the duplicates by deleting the obsolete version of the member or see one of the following sections:
 - “Multiple subsystems with controlled duplicates” on page 269
 - “Separate production MSLs for each developer” on page 272
 - “MSLs that contain unintended duplicates” on page 277
2. Move the parts in the DBA and COMMON MSLs to the sandbox. Use a single concatenation sequence for these two MSLs.
3. Commit the applications created from the DBA and COMMON MSLs to ENVY.
4. Version and release the classes (VAGen part classes and views).

5. Version the applications created from the DBA and COMMON MSLs.
6. Create a configuration map for the ENVY applications created from the DBA and COMMON MSLs.
7. Version this configuration map.
8. Keep the applications created from the DBA and COMMON MSLs in the image and the sandbox. This enables the MSL Migration Assistance Tool to specify these applications as required applications where appropriate when migrating the subsystems.
9. Migrate one of the subsystems (for example, SUBSYS1).
10. Commit this group of applications to ENVY.
11. Version and release the classes (VAGen part classes and views).
12. Version the applications created from the SUBSYS1 MSL.
13. Create a configuration map for the ENVY applications created from the SUBSYS1 MSL. This configuration map should specify as a prerequisite the configuration map created for the applications created from the DBA and COMMON MSLs.
14. Version this configuration map.
15. Package the views and generate the programs moved to ENVY from the SUBSYS1 and COMMON MSLs. Test the code migrated for Subsystem 1.
16. Migrate the next subsystem (SUBSYS2). The ENVY applications created from the DBA, COMMON, and SUBSYS1 MSL can all be available in the image so they can be specified as prerequisites where appropriate.
17. Commit this group of applications to ENVY.
18. Version and release the classes (VAGen part classes and views).
19. Version the applications created from the SUBSYS2 MSL.
20. Create a configuration map for the ENVY applications created from the SUBSYS2 MSL. This configuration map should specify as a prerequisite the configuration map created for the applications created from the DBA and COMMON MSLs.
21. Version this configuration map.
22. Package the views and generate the programs moved to ENVY from the SUBSYS2 MSL. You should not need to generate the programs or package the views in COMMON again unless SUBSYS2 is for a different target environment. Test the code migrated for Subsystem 2.
23. Migrate the work-in-progress MSLs using one of the following techniques:
 - “Complete set of MSLs for production and deltas for test” on page 282
 - “Complete sets of MSLs for test and production” on page 285
 - “MSLs from marketing or other demonstrations” on page 289

Because there are no duplicate parts, the configuration maps for both SUBSYS1 and SUBSYS2 can be loaded into the same ENVY image.

Multiple subsystems with controlled duplicates

The scenario for separate MSLs for each subsystem, but with controlled duplicates has the following MSL structure:

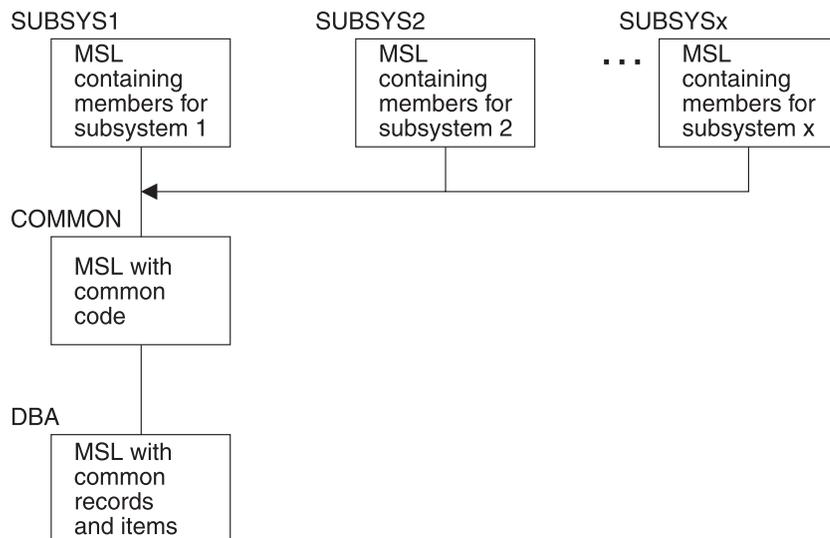


Figure 34. Sample MSL Concatenation for Multiple Subsystems with Controlled Duplicates

Although the scenario in Figure 34 appears the same as the scenario in Figure 33 on page 267, the Figure 34 scenario is different in that the same member might exist in the SUBSYS1, SUBSYS2, and SUBSYSx MSLs. For example, the COMMON MSL might contain a message handling program that obtains the messages from a VisualAge Generator table. Each subsystem has its own VisualAge Generator message table.

This results in duplicates, but only intentional duplicates, between the subsystems. There are no duplicates between a subsystem and the COMMON MSL.

For CICS and IMS target environments, this assumes that the subsystems run in separate regions — duplicate program, table, or map group names are not permitted in the same region.

Recommendations

Consider migrating the MSLs to ENVY as follows:

1. Determine the list of members that have been intentionally duplicated between the subsystems. Ensure that there are no duplicates between the DBA or COMMON MSLs and any of the subsystems.
2. Move the parts in the DBA and COMMON MSLs to the sandbox. Use a single concatenation sequence for these two MSLs. There might be a number of parts marked as *Not Found* because each subsystem contains its own version of the part. Be sure that these parts are included in the subsystem MSLs. For example, if each subsystem has its own message table, the table would not be in either the DBA or COMMON MSL and would be reported as *Not Found* when you migrate these two MSLs.
3. Commit the applications created from the DBA and COMMON MSLs to ENVY.
4. Version and release the classes (VAGen part classes and views).
5. Version the applications created from the DBA and COMMON MSLs.
6. Create a configuration map for the ENVY applications created from the DBA and COMMON MSLs.
7. Version this configuration map.
8. Keep the applications created from the DBA and COMMON MSLs in the image and the sandbox. This enables the MSL Migration Assistance Tool to specify these applications as required applications where appropriate when migrating the subsystems.
9. Migrate one of the subsystems (for example, SUBSYS1).
 - Some of the intentional duplicates might have a **Status** of *Not Found* on the **MSL Migration Part List** window. For example, if an error handling program from COMMON uses different message tables for each subsystem, the message table would be identified as a *Not Found* part until you migrate it to the sandbox for the first subsystem.
 - Put the intentional duplicates for this subsystem into one or more ENVY applications that have a name that identifies them as duplicates. For example:


```
xxxxyyyyyyzzzzzz
```

where:

xxx Is the subsystem ID.

yyyyyy Is something that indicates a duplicate (such as Duplicate, Overwrite, Special, Subsystem).

zzzzzz Is what is being duplicated (such as Messages, MenuTable, HelpText).
 - Mark the applications created for the intentional duplicates as unexplodable.

- Move the other parts from this MSL concatenation to the sandbox.
10. Commit this group of applications to ENVY.
 11. Version and release the classes (VAGen part classes and views).
 12. Version the applications created from the SUBSYS1 MSL.
 13. Create a configuration map for the ENVY applications created from the SUBSYS1 MSL. This configuration map should specify as a prerequisite the configuration map created for the applications created from the DBA and COMMON MSLs. Include the ENVY applications created for this subsystem's intentional duplicates in the configuration map.
 14. Version this configuration map.
 15. Package the views and generate the programs moved to ENVY from the SUBSYS1 and COMMON MSLs. Test the code migrated for Subsystem 1.
 16. Prepare to migrate the next subsystem by doing the following:
 - From the VisualAge Organizer window, unload the applications created for SUBSYS1 from the image. Keep the applications created from DBA and COMMON in the image.
 - From the VG Part Prerequisites View window, delete the applications created for SUBSYS1 from the VG Part Prerequisites View window so that duplicates between Subsystem 1 and Subsystem 2 will not be detected. Before you can delete an application, you must first remove it from the **Required Applications** list of all other applications in the sandbox.
 17. Move the next subsystem (SUBSYS2) to the sandbox. Use the same process that is described in 9 on page 270.
 18. Commit this group of applications to ENVY.
 19. Version and release the classes (VAGen part classes and views).
 20. Version the applications created from the SUBSYS2 MSL.
 21. Create a configuration map for the ENVY applications created from the SUBSYS2 MSL. This configuration map should specify as a prerequisite the configuration map created for the applications created from the DBA and COMMON MSLs.
 22. Version this configuration map.
 23. Package the views and generate the programs moved to ENVY from the SUBSYS2 MSL. If the migration of Subsystem 2 replaced any parts used by the COMMON code, you would need to package the views or generate the programs created from COMMON again. For example, if each subsystem has its own version of Record1, then any program created from COMMON that used Record1 would need to be generated for Subsystem 2. Similarly, any view that uses Record1 would need to be packaged again. Test the code migrated for Subsystem 2.
 24. Migrate the work-in-progress MSLs using one of the following techniques:

- “Complete set of MSLs for production and deltas for test” on page 282
- “Complete sets of MSLs for test and production” on page 285
- “MSLs from marketing or other demonstrations” on page 289

Either the configuration map for SUBSYS1 or for SUBSYS2 can be loaded into an ENVY image, but both cannot be loaded at the same time.

Separate production MSLs for each developer

The scenario for separate production MSLs for each developer has the following MSL structure:

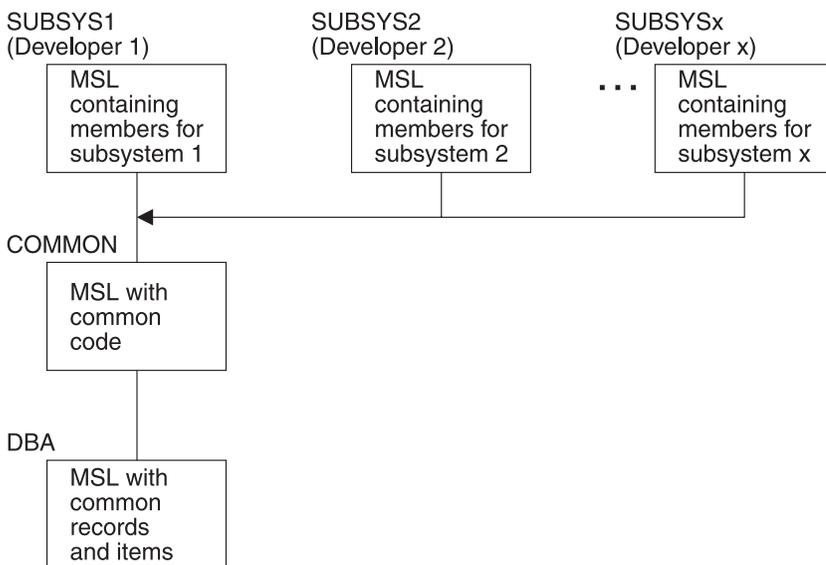


Figure 35. Sample MSL Concatenation when Each Subsystem Replaces Common Code

Although the scenario in Figure 35 appears very similar to Figure 34 on page 269, the situation in Figure 35 is different in that each subsystem was written by a different developer without regard to what the other developers were doing. In this scenario, the same member can exist in the COMMON, SUBSYS1, and SUBSYS2 MSLs. This is because developers are free to modify the common code to suit their own needs. Thus some developers might use the common member unchanged (and still in COMMON) and other developers might each have a different version of the common member. This results in lots of duplicate members, all at different and very likely conflicting levels of code.

There should be a core set of common code, without duplicates in any MSL. However, the following situation could also arise:

SUBSYS1 MSL - members A, B, C	from COMMON: D
SUBSYS2 MSL - members B, C	from COMMON: A, D
SUBSYS3 MSL - members A	from COMMON: B, C, D
SUBSYS4 MSL - members C	from COMMON: A, B, D
COMMON MSL - members A, B, C, D	

In this situation, the COMMON MSL would need to create the following ENVY applications:

```
CommonCore: part D
CommonA:    part A
CommonB:    part B
CommonC:    part C
```

As you can see from this small example, determining how the members could be partitioned efficiently might be quite difficult.

Recommendations

How you migrate when each subsystem has been able to modify the COMMON and DBA code depends on whether you want to define a core set of code that no one is allowed to modify or whether you want each subsystem to have its own version of all code and thus work on a stand-alone basis.

Creating a core common set of applications

If you want to change the philosophy that each developer can modify the common code, consider migrating the MSLs into ENVY as follows:

1. Determine the core set of members from the COMMON and DBA MSLs that none of the developers have modified for any subsystem.
2. Build an external source format file containing only this core set of members.
3. Use the MSL Migration Assistance Tool to create an MSL called CORE for this external source format file.
4. Build a second external source format file containing all the members from COMMON and DBA that are not in the core set (members that are changed in one or more subsystems).
5. Use the MSL Migration Assistance Tool to create an MSL called CHANGED for this external source format file.
6. Move the parts from the CORE MSL to the sandbox. There might be a number of parts marked as *Not Found* because individual subsystems have modified these parts.
7. Commit the applications created from the CORE MSL to ENVY.
8. Version and release all the classes (VAGen part classes and views).
9. Version the applications.
10. Keep the applications created from the CORE MSL in the image and the sandbox. This enables the MSL Migration Assistance Tool to specify these

applications as required applications where appropriate when migrating the subsystems. This also avoids associated parts being marked as *Not Found* if they are part of the core common code.

11. Move the parts for the SUBSYS1 MSL into the sandbox. Include the CHANGED MSL at the bottom of the concatenation sequence. This enables you to locate any associated parts that are not included in the CORE MSL, but which were not modified for SUBSYS1. Do not add or replace any parts in the applications created from the CORE MSL.
12. Commit the applications created from the SUBSYS1 and CHANGED MSLs to ENVY.
13. Version and release all the classes (VAGen part classes and views).
14. Version the applications.
15. Create a configuration map that reflects the current version of the applications. This configuration map represents the parts that existed for Subsystem 1.
16. Version this configuration map.
17. Package the views and generate the programs moved to ENVY for Subsystem 1. Test the code migrated for Subsystem 1.
18. Prepare to migrate the next subsystem by doing the following:
 - From the VisualAge Organizer window, unload the applications created for SUBSYS1 and CHANGED from the image. Keep the applications created from the CORE MSL in the image.
 - From the VG Part Prerequisites View window, delete the applications created for SUBSYS1 and CHANGED from the VG Part Prerequisites View window so that duplicates between Subsystem1 and Subsystem 2 will not be detected. Before you can delete an application, you must first remove it from the **Required Applications** list of all other applications in the sandbox.
19. Move the next subsystem (SUBSYS2) to the sandbox. Include the CHANGED MSL at the bottom of the concatenation sequence. This enables you to locate any associated parts that are not included in the CORE MSL, but which were not modified for SUBSYS2.

Be sure the applications created from the CORE MSL are loaded into the image and the sandbox. This avoids associated parts being marked as *Not Found* if they are part of the core common code. Do not add or replace any parts in the applications created from the CORE MSL.
20. Commit the applications created from the SUBSYS2 and CHANGED MSLs to ENVY.
21. Version and release all the classes (VAGen part classes and views).
22. Version the applications.

23. Create a configuration map that reflects the current version of the applications. This configuration map represents the parts that existed for Subsystem 2.
24. Version this configuration map.
25. Package the views and generate the programs moved to ENVY for Subsystem 2. If the migration of Subsystem 2 replaced any parts used by the CORE code, you would need to package the views or generate the programs created from CORE again. For example, if each subsystem has its own version of Record1, then any program created from CORE that used Record1 would need to be generated for Subsystem 2. Similarly, any view that uses Record1 would need to be packaged again. Test the code migrated for Subsystem 2.
26. Migrate the work-in-progress MSLs using one of the following techniques:
 - “Complete set of MSLs for production and deltas for test” on page 282
 - “Complete sets of MSLs for test and production” on page 285
 - “MSLs from marketing or other demonstrations” on page 289

Creating stand-alone subsystems

If you want to each subsystem to have its own version of all code so its developers can work on a stand-alone basis, consider migrating your MSLs to ENVY as follows:

1. For the first subsystem, move parts to the sandbox using a concatenation sequence of SUBSYS1, COMMON, and DBA. The concatenation sequence should be the same concatenation sequence that you use to generate programs for Subsystem 1.
 If there are parts in COMMON and DBA that are not currently used by Subsystem 1, consider putting them into an application called xxxUnusedPartsApp, where xxx is the subsystem ID. Using this technique means that the common definition of these unused parts will be available in the set of applications for Subsystem 1.
2. Commit the applications created for Subsystem 1 to ENVY. If you think that Subsystem 1 might need parts from xxxUnusedPartsApp in the future, commit it to ENVY. The other subsystems will develop their own lists of unused parts.
3. Version and release all the classes (VAGen part classes and views).
4. Version the applications.
5. Create a configuration map that reflects the current version of the applications. This configuration map represents the parts that existed for Subsystem 1.

If you created an xxxUnusedPartsApp, you can include it in the configuration map if you think the parts are likely to be used in the

future or omit it from the configuration map if there are a large number of parts or you do not expect them to be used in the future.

6. Version this configuration map.
7. Package the views and generate the programs moved to ENVY for Subsystem 1. Test the code migrated for Subsystem 1.
8. Prepare to migrate the next subsystem by doing the following:
 - From the VisualAge Organizer window, unload all the applications created for Subsystem 1 from the image.
 - From the VG Part Prerequisites View window, delete all the applications created for Subsystem 1, including the applications created from the COMMON or DBA MSLs and the xxxUnusedPartsApp
9. Move the next subsystem to the sandbox using a concatenation sequence of SUBSYS2, COMMON, and DBA. The concatenation sequence should be the same concatenation sequence that you use to generate programs for Subsystem 2.

Note: This technique means that identical code for a part might exist in the applications created for Subsystem 1 and Subsystem 2. This occurs if both of these subsystems used the version of the part that was previously in COMMON or DBA. However, some other subsystem might have modified the part or the developers for Subsystem 1 might need to have their own version of the part in the future. This technique enables the developers of Subsystem 1 and Subsystem 2 to continue to develop in an independent manner, without regard to how the other subsystem uses the part.

10. Commit the applications created for Subsystem 2 to ENVY.
11. Version and release all the classes (VAGen part classes and views).
12. Version the applications.
13. Create a configuration map that reflects the current version of the applications. This configuration map represents the parts that existed for Subsystem 2.
14. Version this configuration map.
15. Package the views and generate the programs moved to ENVY for Subsystem 2. Because each subsystem has its own complete set of code, you should generate all the programs and package all the views for Subsystem 2. Test the code migrated for Subsystem 2.
16. Migrate the work-in-progress MSLs using one of the following techniques:
 - “Complete set of MSLs for production and deltas for test” on page 282
 - “Complete sets of MSLs for test and production” on page 285
 - “MSLs from marketing or other demonstrations” on page 289

MSLs that contain unintended duplicates

This scenario contains unintended duplicates. There are multiple subsystems, with the following **intended** MSL structure:

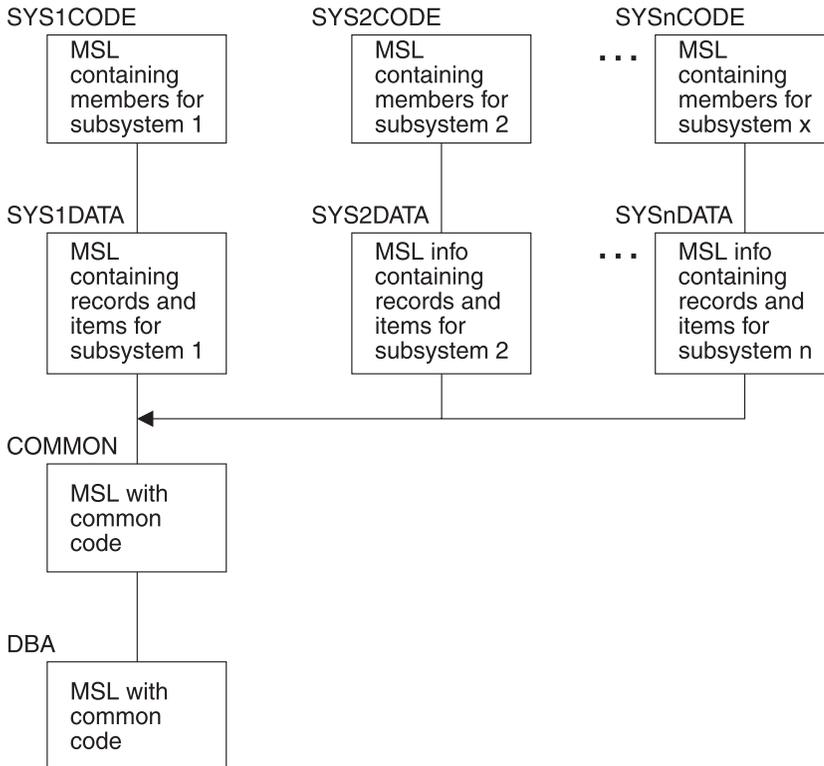


Figure 36. Sample MSL Concatenation when Using VisualAge Generator Templates

Each member was intended to be in one and only one MSL. However, the MSL structure contained the following:

- Some members were never advanced to the production MSLs; they were still in the developer's read/write MSLs even though the changes were complete and were in the production load libraries.
- Some members that were shared between subsystems were not contained in the COMMON MSL; they were in only one subsystem's MSLs. For example, this occurred for records and their associated global data items that were passed when transferring between two subsystems. In this case the real concatenation sequence was SYS1CODE, SYS1DATA, SYS2DATA, and COMMON. In some cases several additional data MSLs were required to generate a subsystem.

- Applications, processes, statement groups, and maps existed both in the SYSnCODE and SYSnDATA MSL — with the same names. This was because they were accidentally put into the wrong MSLs.
- The same members existed in multiple subsystems. For example, the same process might appear in SYS1CODE, SYS3CODE and SYSnCODE.
- Two subsystems used a different COMMON MSL, called COMMON2. COMMON2 contained 10 - 20 programs, as well as other members, that were also included in COMMON.

Recommendations

Before attempting to migrate this set of MSLs, do the following:

- Be sure you know the real MSL concatenation sequences that are required for generation. This will help to minimize the number of *Not Found* members during migration. One possibility is to validate existing applications in the MSLs prior to attempting migration to determine which, if any, members are missing.
- Try to resolve as many duplicates as possible. One possibility is to use the MSL Migration Assistance Tool for a trial migration to determine what duplicates exist and which members are missing. Then go back to the MSLs, make any corrections required to resolve duplicates, restore the missing members, and correct any other errors in the parts.
- After you have resolved the problems, then migrate using the scenario that best matches your new environment.

MSLs containing code from VisualAge Generator Templates or BW*Wizard

The scenario for VisualAge Generator Templates or BW*Wizard has the following MSL structure:

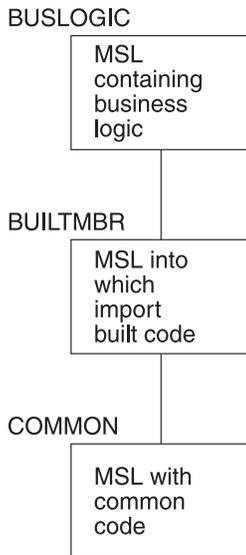


Figure 37. Sample MSL Concatenation when Using VisualAge Generator Templates or BW*Wizard

Programs built with VisualAge Generator Templates or BW*Wizard are imported into the MSL called BUILTMBR. Then the BUSLOGIC read/write MSL is concatenated in front of the read-only BUILTMBR MSL and all *business logic* changes are made in BUSLOGIC. The advantages of this technique are:

- The program can be built again from the template and imported into the BUILTMBR MSL without destroying the business logic.
- If necessary, a comparison of the newly built member in BUILTMBR with the old business logic version of the member can be done to determine if any business logic changes are required to incorporate function that has been added to the newly built member.

This same pairing of a business logic MSL and “built member” MSL can be extended from the developer’s MSLs to the staging, test, and production MSLs. In some cases, the separation of the business logic might be used for the developer’s, staging, and test MSLs, but not be used for the production MSLs.

The COMMON MSL contains any members that are used across subsystems. There might also be a DBA MSL that contains data item definitions.

This example assumes that all the code in BUILTMBR and BUSLOGIC are for a single subsystem called Subsystem 1. If you have multiple subsystems, you would need to extrapolate the example for your situation.

Recommendations

Consider migrating the MSLs to ENVY as follows:

1. Move the parts from the COMMON MSL to the sandbox.

Note: If there is a DBA MSL, migrate it with the COMMON MSL in a single concatenation sequence.

2. Commit the applications created from the COMMON MSL (and DBA MSL if used) to ENVY.
3. Version and release all the classes (VAGen part classes and views). You can use **1.0** (the default) as the version number.
4. Version the applications. You can use **1.0** (the default) as the version number.
5. Create a configuration map that contains the applications created from the COMMON MSL. If there was a DBA MSL, also include the applications created from the DBA MSL.
6. Version this configuration map.
7. Keep the applications created from the COMMON MSL (and DBA MSL if used) in the image and the sandbox. This enables the MSL Migration Assistance Tool to specify these applications as required applications where appropriate when migrating the subsystems.
8. Move parts from the BUILTMBR MSL to the sandbox.
9. Commit the applications created from the BUILTMBR MSL to ENVY.
10. Version and release all the classes (VAGen part classes and views). Name the version to indicate that it is for parts built by the templates. For example, you could use `Built-1.0`.
11. Version the applications. Name the version to indicate it is for parts built by the templates. For example, you could use `Built-1.0`. Versioning the application enables you to reload the level of code that was built by the template without having to load each class individually.
12. If you might frequently need to load the level of code built from the templates, create a configuration map that reflects the current version of the applications. Version this configuration map, using a version of `Built-1.0`.
13. Keep all the applications from the COMMON and BUILTMBR MSLs in the sandbox and in your image. This enables you to replace parts in the applications created from the BUILTMBR MSL.
14. Move the parts in the BUSLOGIC MSL to the sandbox. You might have two types of parts in the business logic MSL:
 - Parts that you added for business logic that were not originally built by the templates. These parts will have blanks in the **Duplicate** and **Last Migration Library Timestamp** columns. You can create new applications for these parts or add them into an existing (committed)

application. If you add or change parts in an existing application, the MSL Migration Assistance Tool marks the application as *Modified* and you will be able to commit the application to ENVY again to put the new parts into the ENVY library manager.

- Parts that were originally built for the templates that you modified for business logic. These parts will have *True* in the **Duplicate** column and the timestamp from the template-built MSL in the **Last Migration Library Timestamp** column. Move these parts to the sandbox using **Handle Duplicate Parts** and specifying that they should **Replace Existing** parts. This causes each part to be placed in the same application where it already exists. The MSL Migration Assistance Tool marks the application as *Modified* and you will be able to commit the application to ENVY again to put the new parts into the ENVY library manager.

This technique preserves the application organization you created when you migrated the BUILTMBR MSL, but to include the business logic version of existing parts and any new parts that were not created by the templates.

Note: If you do not have any new parts created for business logic in the BUSLOGIC MSL, you can:

- Create an external source format file for the BUSLOGIC MSL.
 - Use **VAGen Import** to import the external source format file and create new editions of the changed parts. Select the **Defined application** radio button to put the changed editions of the parts into the applications in which they are already defined.
15. Commit any applications that were created or modified for the BUSLOGIC MSL to ENVY.
 16. Version and release all the classes (VAGen part classes and views). Name the version to indicate that it is for parts changed for business logic. For example, you could use `BusLogic-1.1`.
 17. Version the applications. Name the version to indicate it is for parts changed for business logic. For example, you could use `BusLogic-1.1`.

Note: Not all of the classes or applications will have new versions created when you migrate the BUSLOGIC MSL. Only those classes and applications modified for business logic will have a new version.

18. Create a configuration map that reflects the current version of the applications. This configuration map represents the actual system that was contained in the concatenation sequence for the BUSLOGIC, BUILTMBR, and COMMON MSLs.

This configuration map should include the BusLogic-1.1 version of the applications created from the BUILTMBR and BUSLOGIC MSLs. This configuration map should specify as a required map the configuration map for the applications created from the COMMON (and DBA) MSL.

19. Version this configuration map using version BusLogic-1.1.
20. Package the views and generate the programs moved to ENVY. Test the code.
21. Migrate the work-in-progress MSLs using one of the following techniques:
 - “Complete set of MSLs for production and deltas for test”
 - “Complete sets of MSLs for test and production” on page 285
 - “MSLs from marketing or other demonstrations” on page 289

You will need to adapt these techniques to capture both the parts built from the templates and the parts that have business logic.

Special considerations for VisualAge Generator Templates

Make sure you change all Statement Group part names to uppercase before migration. The VisualAge Generator Templates product generates Statement Group part names as either all lowercase or mixed case. However, VisualAge Generator searches for these part names in all uppercase (when generating runtime code) and cannot find the parts when the names are in mixed case or lowercase.

Complete set of MSLs for production and deltas for test

This scenario has the following MSL structure. The Database Administrator (DBA) MSLs might be combined with the corresponding STAGING, SYSTEST, or PROD MSL depending on your environment.

Note: There might be one series of the MSLs shown in Figure 38 on page 283 for each subsystem.

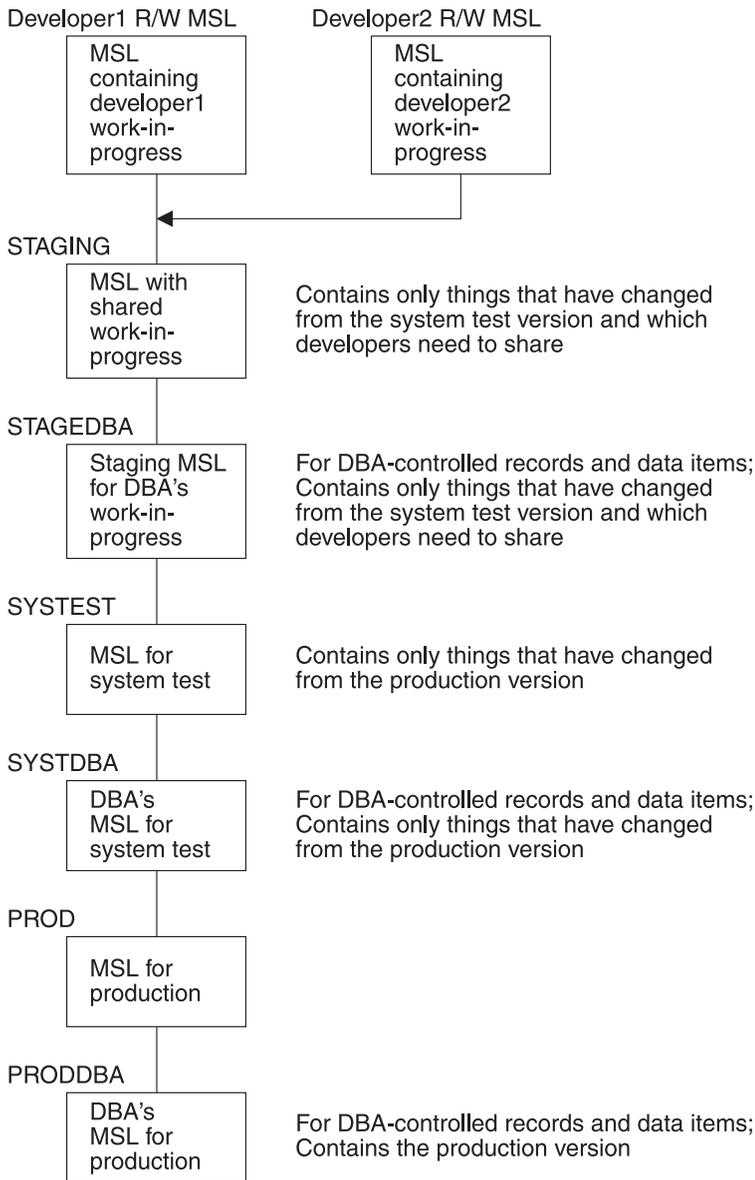


Figure 38. Sample MSL Concatenation for Complete Production MSLs and Deltas for Test

The PROD and PRODDBA MSLs are the only complete pair of MSLs. In other words, this is the only pair of MSLs in which all members exist. SYSTEST and SYSTDBA contain only the members that have been added or changed and which are undergoing system test. STAGING and STAGEDBA contain only the members that have been added or changed and which are still in development testing, but which need to be shared by multiple developers.

Two additional MSL concatenation sequences are used for advancing (moving) members through the MSL hierarchy.

DEVnMSL+STAGING+SYSTEST+PROD - to advance members not controlled by DBA

DEVDBA+STAGEDBA+SYSTDBA+PRODDBA - to advance members controlled by DBA

As newly added or changed members progress through testing, they advance from the developer's read/write MSL to the staging MSL, then from staging to the appropriate development level MSL, then from development to the corresponding system test MSL, and finally from system test to the corresponding production level MSL. Thus the developer read/write, staging, and system test MSLs represent work-in-progress. Similarly, the DEVDBA, STAGEDBA, and SYSTDBA MSLs represent work-in-progress for the DBA.

If there are multiple subsystems, each has its own staging, system test, and production MSLs. There are no duplicate members between the subsystems.

Recommendations

Consider migrating the MSLs to ENVY as follows:

1. Migrate the production MSLs by doing the following:
 - Load the production level MSLs using the MSL Migration Assistance Tool.
 - Create a configuration map for the production level code.
 - Version this configuration map.
 - Package the views and generate the programs. Then test to be sure that what you migrated matches your production code.
2. Migrate the system test MSLs by doing the following:
 - Open new editions of the applications.
 - Export an external source format file for the SYSTEST MSL and another external source format file for the SYSTDBA MSL.
 - Use **VAGen Import** to import the external source format files and create new editions of the changed parts. Select the **Defined application** radio button to put the changed editions of the parts into the application(s) in which they are already defined.
 - Version and release the changed classes (views and VAGen part classes).
 - Version any ENVY applications that had new editions of classes.
 - Create a configuration map for the system test level of code.
 - Version this configuration map.
 - Package the views and generate the programs for the system test level of code. You should only need to package the views and generate programs, map groups, and tables that have changed. Then test to be sure that what you migrated matches your system test level of code.

3. Migrate the staging MSLs by doing the following:
 - Open new editions of the applications.
 - Export an external source format file for the STAGING MSL and another external source format file for the STAGEDBA MSL.
 - Use **VAGen Import** to import the external source format files and create new editions of the changed parts. Select the **Defined application** radio button to put the changed editions of the parts into the applications in which they are already defined.
 - Version and release the changed classes (views and VAGen part classes).
 - Version any ENVY applications that had new editions of classes.
 - Create a configuration map for the staging level of code.
 - Version this configuration map.
 - Package the views and generate the programs for the staging level of code. You should only need to package the views and generate programs, map groups, and tables that have changed. Then test to be sure that what you migrated matches your staging level of code.
4. Assign ownership of the classes and assign managers for the applications and configuration maps.
5. Open new editions of the applications.
6. Developers can then load their own work-in-progress doing the following:
 - Export an external source format file for their own read/write MSL.
 - Load the configuration map for the staging level of code into their image.
 - Use **VAGen Import** to import the external source format files and create new editions of the changed parts. Select the **Defined application** radio button to put the changed editions of the parts into the applications in which they are already defined.

Note that this scenario is easier to load into ENVY than “Complete sets of MSLs for test and production” because the added/changed members for the system test MSL are already determined (everything in the SYSTEST MSL) and because there is no development level set of MSLs to load.

Complete sets of MSLs for test and production

This scenario has the following MSL structure:

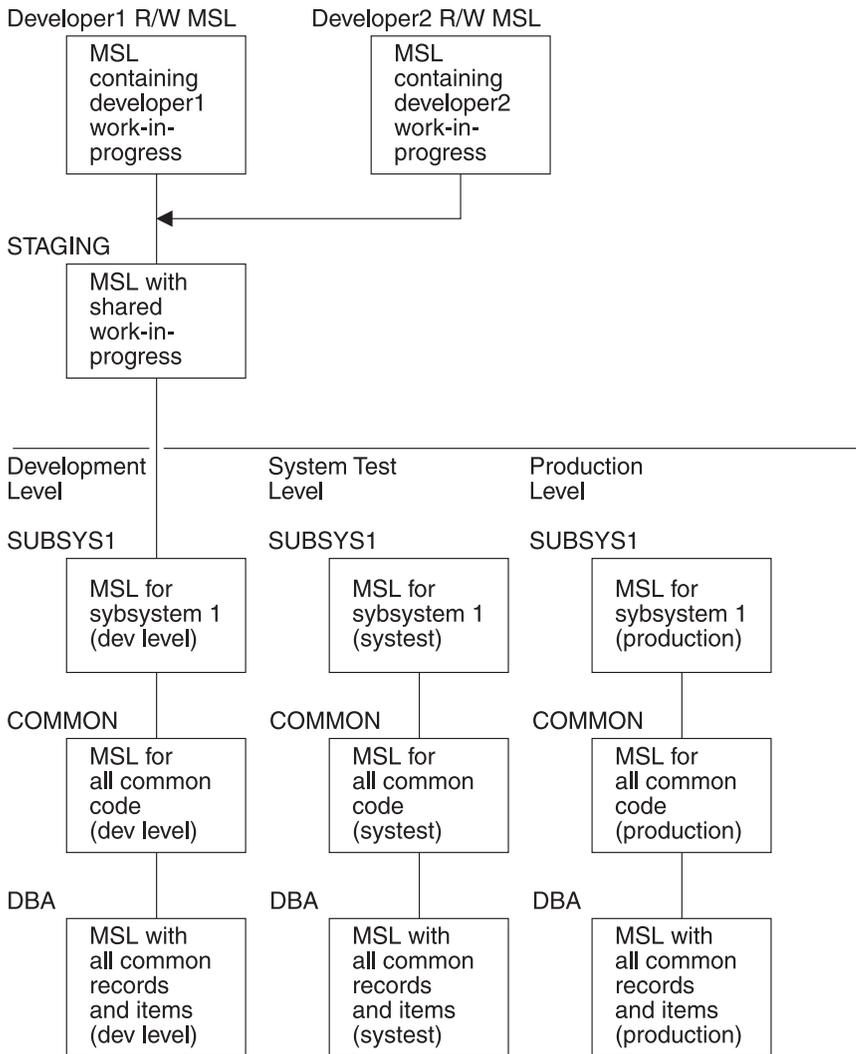


Figure 39. Sample MSL Concatenation for Complete Sets of Development, System Test, and Production

This scenario differs from that described in “Complete set of MSLs for production and deltas for test” on page 282 in that in Figure 39 the SUBSYS1, COMMON, and DBA MSLs at each of the development, system test, and production levels are complete sets of MSLs. In other words, most members exist in all three MSL concatenation sequences with the same date/time stamp. If a member is in the COMMON development MSL, it is also in the COMMON system test MSL, and the COMMON production MSL.

As members progress through testing, the added or changed members advance from the developer’s read/write MSL to the staging MSL and then

from staging to the appropriate development level MSL. As further testing occurs, members are **copied (not moved or advanced)** from the development level MSL to the corresponding system test MSL and then from the system test MSL to the corresponding production MSL.

The developer read/write, staging, development level MSLs, and system test MSLs represent work-in-progress. However, the development level MSLs and the system test MSLs are not true work-in-progress. For example, the system test SUBSYS1 MSL contains the same members as the production SUBSYS1 MSL. Most of the members are identical between system test and production. Perhaps only 5% of the members in the system test SUBSYS1 MSL are currently undergoing system test and represent different versions of the members from what is in the production SUBSYS1 MSL. Similarly, the development level SUBSYS1 MSL for the most part duplicates the members in the system test and production SUBSYS1 MSLs. Perhaps another 5% of the members differ from what is in the system test MSL and represent work that is in development testing (not yet put into system test).

In the same way, the COMMON and DBA MSLs are virtually identical at the development, system test, and production levels. Because the members in these MSLs change less frequently, in many cases there might be **no** members that differ between the three levels of these MSLs.

This means that there are many members in the development level and system test MSLs that should not really be put into ENVY because they are no different than the members that will be migrated using the production MSLs.

Note: If there are multiple subsystems, they can share the COMMON and DBA MSLs, but each subsystem has its own SUBSYSn MSL, where n is the subsystem number. The SUBSYSn MSLs do not have any duplicate member names.

Recommendations

Consider migrating the MSLs to ENVY as follows:

1. Migrate the production MSLs by doing the following:
 - Load the production level MSLs using the MSL Migration Assistance Tool.
 - Create a configuration map for the production level code.
 - Version this configuration map.
 - Package the views and generate the programs. Then test to be sure that what you migrated matches your production code.
2. Migrate the system test MSLs by doing the following:
 - Open new editions of the applications.

- Determine the list of members that have been added/changed in system test by eliminating the members that match production based on the date/time stamp.
 - Export an external source format file that contains only the members that actually changed between the system test and production versions of the SUBSYS1 MSL, a second external source format file for the changes between the system test and production versions of the COMMON MSL, and a third external source format file for the changes between the system test and production versions of the DBA MSL.
 - Use **VAGen Import** to import the external source format files and create new editions of the changed parts. Select the **Defined application** radio button to put the changed editions of the parts into the applications in which they are already defined.
 - Version and release the changed classes (views and VAGen part classes).
 - Version any ENVY applications that had new editions of classes.
 - Create a configuration map for the system test level of code.
 - Version this configuration map.
 - Package the views and generate the programs for the system test level of code. You should only need to package the views and generate programs, map groups, and tables that have changed. Then test to be sure that what you migrated matches your system test level of code.
3. Migrate the development level MSLs by doing the following:
- Open new editions of the applications.
 - Determine the list of members that have been added/changed in development by eliminating the members that match system test based on the date/time stamp.
 - Export an external source format file that contains only the members that actually changed between the development and system test versions of the SUBSYS1 MSL, a second external source format file for the changes between the development and system test versions of the COMMON MSL, and a third external source format file for the changes between the development and system test versions of the DBA MSL.
 - Use **VAGen Import** to import the external source format files and create new editions of the changed parts. Select the **Defined application** radio button to put the changed editions of the parts into the applications in which they are already defined.
 - Version and release the changed classes (views and VAGen part classes).
 - Version any ENVY applications that had new editions of classes.
 - Create a configuration map for the development level of code.
 - Version this configuration map.
 - Package the views and generate the programs for the development level of code. You should only need to package the views and generate

programs, map groups, and tables that have changed. Then test to be sure that what you migrated matches your development level of code.

4. Migrate the staging MSLs by doing the following:
 - Open new editions of the applications.
 - Determine the list of members that have been added/changed in the staging MSL by eliminating the members that match development based on the date/time stamp.
 - Export an external source format file that contains only the members that actually changed between the STAGING MSL and development MSLs.
 - Use **VAGen Import** to import the external source format file and create new editions of the changed parts. Select the **Defined application** radio button to put the changed editions of the parts into the applications in which they are already defined.
 - Version and release the changed classes (views and VAGen part classes).
 - Version any ENVY applications that had new editions of classes.
 - Create a configuration map for the staging level of code.
 - Version this configuration map.
 - Package the views and generate the programs for the staging level of code. You should only need to package the views and generate programs, map groups, and tables that have changed. Then test to be sure that what you migrated matches your staging level of code.
5. Assign ownership of the classes and assign managers for the applications and configuration maps.
6. Open new editions of the applications.
7. Developers can then load their own work-in-progress doing the following:
 - Export an external source format file for their own read/write MSL.
 - Load the configuration map for the staging level of code into their image.
 - Use **VAGen Import** to import the external source format files and create new editions of the changed parts. Select the **Defined application** radio button to put the changed editions of the parts into the applications in which they are already defined.

MSLs from marketing or other demonstrations

The scenario for MSLs used in demonstrations reflects the need to have snapshots of the same MSL from various stages of development.

ENDDEMO

MSL
containing
members
built by
end of
demonstration

MIDDEMO

MSL
containing
members
built by
middle of
demonstration

STRTDEMO

MSL
containing
members
at start of
demonstration

Figure 40. Sample MSLs for a Demonstration

The three MSLs are used as follows:

- STRTDEMO is the MSL at the start of the demonstration. It might contain some partially developed members, as well as a completed server program.
- MIDDEMO contains members that are added or changed during the first part of the demonstration.
- ENDDEMO represents members that are added or changed during the entire demonstration.

This technique enables the person doing the demonstration to quickly reset the demonstration to specific scenarios.

Recommendations

Using a single ENVY application

For a small demonstration, for which all the parts can be stored into a single ENVY application, consider migrating as follows:

1. Create an external source format file for each of the three MSLs.
2. Using the VisualAge Organizer window, create a new application.
3. Use **VAGen Import** to import the external source format file for the STRTDEMO MSL.

4. Version and release the classes (views and VAGen parts).
5. Version the application.
6. Create a new edition of the application.
7. Use **VAGen Import** to import the external source format file for the MIDDEMO MSL.
8. Version and release the classes (views and VAGen parts).
9. Version the application.
10. Create a new edition of the application.
11. Use **VAGen Import** to import the external source format file for the ENDDEMO MSL.
12. Version and release the classes (views and VAGen parts).
13. Version the application.

You can now reset the demonstration to specific points by loading the correct version of the application.

Using multiple ENVY applications

For a larger demonstration, for which the parts need to be separated into several ENVY applications, consider migrating as follows:

1. Create an external source format file for each of the three restructured MSLs.
2. Using the MSL Migration Assistance Tool load the parts from the STRTDEMO MSL, separating them into ENVY applications as needed for your demonstration. Do not be concerned about missing parts because you know that these will be developed during the demonstration.
3. Version and release the classes (views and VAGen parts).
4. Version the applications.
5. Create a configuration map that represents the starting point for the demonstration.
6. Version this configuration map.
7. Create a new edition for each of the applications.
8. Create any new applications that are needed to contain parts that are added during the first part of the demonstration.
9. Use **VAGen Import** to import the external source format file for the MIDDEMO MSL. For new parts, move each part to the corresponding application. For existing parts, use the **Defined application** radio button to put the changes into the applications in which the parts are already defined.
10. Version and release the classes (views and VAGen parts).
11. Version the applications.

12. Open a new edition of the configuration map and modify it so that it contains the versions of applications that are added or changed during the first part of the demonstration.
13. Version this configuration map.
14. Create a new edition of each of the applications.
15. Create any new applications that are needed to contain parts that are added during the second part of the demonstration.
16. Use **VAGen Import** to import the external source format file for the ENDDemo MSL. For new parts, move each part to the corresponding application. For existing parts, use the **Defined application** radio button to put the changes into the applications in which the parts are already defined.
17. Version and release the classes (views and VAGen parts).
18. Version the applications.
19. Open a new edition of the configuration map and modify it so that it contains the versions of applications that are added or changed during the first part of the demonstration.
20. Version this configuration map.

You can now reset the demonstration to specific points by loading the correct version of the configuration map.

Chapter 29. Running the MSL Migration Assistance Tool on Smalltalk

The sections that follow describe tasks that you will need to do during migration, including:

- Getting ready to migrate:
 - “Starting VisualAge Generator”.
 - “Creating users and setting the current user” on page 294.
 - “Loading a feature” on page 295.
 - “Collecting your source code” on page 296.
 - “Handling code page changes” on page 297.
 - “Starting the MSL Migration Assistance Tool” on page 300.
 - “Building MSL directories” on page 300.
 - “Resetting the sandbox from ENVY” on page 302.
 - “Selecting your MSLs” on page 304.
- Moving parts to the sandbox and working with the sandbox:
 - “Selecting and migrating VAGen parts” on page 305.
 - “Creating a new application” on page 308.
 - “Moving a VAGen part between applications” on page 308.
 - “Controlling the creation of ApplicationNodes” on page 310.
 - “Renaming an application” on page 311.
 - “Collapsing an application” on page 311.
 - “Handling Duplicates” on page 312.
 - “Finding the application in which a part is located” on page 316.
 - “Listing missing (not found) parts” on page 317.
 - “Handling missing (not found) parts” on page 318.
 - “Checking relationships among applications” on page 319.
 - “Updating the list of required applications” on page 321.
 - “Deleting an ApplicationNode” on page 323.
 - “Deleting an application” on page 324.
- “Committing to ENVY” on page 325.

Note: ENVY provides a variety of ways of doing most tasks and supports multiple development scenarios. This chapter is intended to provide a way, but not necessarily the only way, of performing tasks related to migrating MSLs to ENVY.

Starting VisualAge Generator

This section describes how to start VisualAge Generator Version 4.0, create user IDs, set the user ID for the current user, and then import and load the MSL Migration Assistance Tool.

Action	Description
<p>Start VisualAge Generator from either the VisualAge Smalltalk or VisualAge Generator icon.</p> <p>Alternatively, from an OS/2 window or Windows NT command prompt, change to the directory where VisualAge Generator 4.0 is installed and type:</p> <pre>start abt</pre>	<p>The System Transcript window is displayed, then the Selection Required window is displayed prompting for the name of the user who owns this image.</p>
<p>Select <i>Library Supervisor</i> from the list of users.</p>	<p>This enables you to define other users before running the MSL Migration Assistance Tool.</p>
<p>When the message saying “You must connect the image to the current library” is displayed, select the OK push button.</p>	
<p>When the message saying “Done connecting image to the library” is displayed, select the OK push button.</p>	<p>The VisualAge Organizer window is displayed.</p>
<p>From the VisualAge Organizer window, select Options and then Full Menus.</p>	<p>Selecting full menus enables you to see all the options, including the options to change the owner of a class or the manager of an application.</p>
<p>From the VisualAge Organizer window, select Options and then Preferences.</p>	<p>The VisualAge Preferences window is displayed.</p>
<p>On the General page, consider changing the <i>Preferred Settings View</i> to Properties Table (Recommended). Do not make this change if some of the tools you use with VisualAge Smalltalk do not yet support properties tables.</p>	<p>The examples in “Tasks you must do after migrating GUIs” on page 234 use the properties tables in explaining changes you might need to make to GUIs after they are migrated to views.</p>
<p>Select the OK push button to close the VisualAge Preferences window.</p>	<p>The VisualAge Organizer window is displayed.</p>

Creating users and setting the current user

You must be the Library Supervisor to create new users.

Action	Description
<p>From the VisualAge Organizer window, select Options and then Users.</p>	<p>The Users window is displayed listing the available users. The <i>Current user</i> is displayed on the lower left side above the push buttons.</p>
<p>Select the Add push button.</p>	<p>The Users Dialog window is displayed.</p>

Action	Description
Specify the following:	The Users window is displayed with the new user added.
Unique Name	This could be the user's logon ID or employee number.
Full Name	This is the first and last name for the person.
Network Name	This is the user's network logon ID.
Select the OK push button.	
Repeat the above steps to define all of your developers.	
From the Users window, select the user that corresponds to the team leader and select the Change User push button.	A Question window is displayed asking you to confirm that you want to change to this user.
Select the Yes push button to confirm you want to change to the new user.	<p>The Users window is displayed and shows the team leader as the <i>Current user</i> on the lower left side.</p> <p>Setting the user ID for the image to the team leader means that the team leader becomes the class owner and application manager for all the views, VAGen part classes, and applications that are created with the MSL Migration Assistance Tool. After migration, the team leader can assign ownership to the individual developers if needed.</p>
Select the OK push button.	The VisualAge Organizer window is displayed and indicates in the title bar that the team leader is now the owner of the image.

Loading a feature

You might need to load the VisualAge Smalltalk Multimedia or DDE Support features before you migrate your GUIs.

Note: If you use the multimedia feature, refer to the VisualAge Smalltalk installation documentation and readme file for information on how to obtain this feature.

Action	Description
From the VisualAge Organizer window, select Options and then Load/Unload Features .	A Selection Required window appears.
The two features you might need to migrate your GUIs are <i>VisualAge: Multimedia</i> and <i>VisualAge: DDE Support</i> .	A progress window appears indicating the status of the load. The load might take 5 to 10 minutes.
Select each feature in the left pane and then select >> to move the feature to the right pane.	After the load completes, another progress window appears indicating the status for <i>recaching the pointers</i> .
When you have moved both features to the right pane, select the OK push button.	After recaching, a Question window appears prompting you to save your image. Select the OK push button and save the image.
Check the System Transcript window for any error messages.	

Collecting your source code

You need to collect your source code before you can use the MSL Migration Assistance Tool. The steps necessary to do this vary depending on the environment from which you are migrating.

From Cross System Product

If you are migrating from Cross System Product, you must create an external source format file for each of your MSLs and then use the MSL Migration Assistance Tool to create an MSL directory structure. Before downloading all your external source format files, be sure to test your download program to ensure that special characters are converted correctly. You might need to define a conversion table for the download program. Also review “Using the HPTRULES.NLS file” on page 297 for information on handling the not sign (-).

From VisualAge Generator with TeamConnection and no MSLs

If you are migrating from a previous release of VisualAge Generator and use TeamConnection, you might not have MSLs. In this case you must create external source format files for your parts in TeamConnection and then use the MSL Migration Assistance Tool to create an MSL directory structure. You might want to create an external source format file for each component to help you in preserving your existing organizational structure.

In addition, if you are changing from developing on OS/2 to developing on Windows NT, be sure to review “Using the HPTRULES.NLS file” on page 297 and “Changing from OS/2 to Windows NT” on page 298.

From VisualAge Generator MSLs

If you already have VisualAge Generator MSLs and plan to remain on the OS/2 development platform, you can skip this step. The MSL Migration Assistance Tool can use your existing MSLs.

If you are changing from developing on OS/2 to developing on Windows NT, be sure to review “Using the HPTRULES.NLS file” and “Changing from OS/2 to Windows NT” on page 298.

Handling code page changes

If you are migrating from Cross System Product, the code page on the host is EBCDIC and the code page on the workstation is ASCII. You should review “Using the HPTRULES.NLS file”.

If you are migrating from VisualAge Generator and changing from an OS/2 to a Windows NT development platform, the code pages for SBCS languages are different. You should review both “Using the HPTRULES.NLS file” and “Changing from OS/2 to Windows NT” on page 298.

If you are migrating from VisualAge Generator and use a DBCS language, you can skip this section because the code pages are the same for OS/2 and Windows NT.

Using the HPTRULES.NLS file

The not sign used for Cross System Product is the \neg . However, some download programs convert \neg to $\frac{3}{4}$. For VisualAge Generator, the code point for \neg differs between the OS/2 and Windows NT code pages.

The *hptrules.nls* file in the `\vast\nls` directory enables you to specify national language information and can help with handling the not sign. One section of the file enables you to enter your three national language characters, the $\hat{\neg}$ for the not sign, and an alternate not sign. If you determine that the only special character that you need to convert is the \neg or $\frac{3}{4}$, you can do the following to handle the conversion:

1. Shut down VisualAge Generator Developer.
2. Edit *hptrules.nls*
3. Read the comments in the file. The section you need to change is the first section of the file, called `:nlsrules`.
4. In the `:nlsrules` section, there are three columns: the locale, 5 special characters, and the default language code.

The five special characters are:

Column	Description
--------	-------------

- 1 - 3 The three national language characters (\$#@ for English-US).
- 4 The ^ which is the standard not sign for VisualAge Generator.
- 5 An alternate not sign, which you can set to ˘, ¼, or whatever character your download program turned your not sign into.

5. Bring VisualAge Generator Developer back up.

When you use the **ESF to MSL** function (described in “Building MSL directories” on page 300), the MSL Migration Assistance Tool converts any occurrence of the alternate not sign specified in *hptrules.nls* to the ^ . This conversion only applies to occurrences in processes, statement groups, or functions. The constant delimiter for a map is not converted, but because the constant delimiter is stored with each map, conversion is not necessary. Conversion of the alternate not sign also happens when you use **VAGen Import**. Conversion does not occur when you save changes to functions. Therefore, you should use the ^ for any new development work.

If your only code page conversion problems (whether from downloading Cross System Product code or from moving external source format files from OS/2 to Windows NT) are due to your not sign, you should be able to manipulate the *hptrules.nls* file to handle the conversion for you.

Changing from OS/2 to Windows NT

Note: If you are migrating from Cross System Product, or if you are migrating from VisualAge Generator and use a DBCS code page, skip this section. Code page conversion is not required in these situations.

The code pages from some languages differ between OS/2 and Windows NT. For example, the code point for the ˘ is different for the English-US code pages for OS/2 and Windows NT. Therefore, you must convert your VisualAge Generator code if you are moving from the OS/2 development environment to the Windows NT environment. To convert between the code pages, do the following:

1. If you are migrating from VisualAge Generator 3.0 or 3.1 on OS/2, create an external source format file for each ENVY application. When migrating from any other VisualAge Generator release on OS/2, create an external source format file for each MSL. Be sure to export external source format, not binary, for your GUIs.
2. Make the external source format file available to Windows NT.
3. From Windows NT, change to the directory in which *hptcnoXY.exe* is located. (Substitute the number of your VisualAge Generator version for

the X and your release for the Y.) If you used the default directories when you installed, the directory will be c:\vast.

4. Convert the external source format file by running the following:

```
hptcnvXY esf-file-name conversion-table
```

where:

X Is the number of your version of VisualAge Generator — for example, use a **4** for VisualAge Generator 4.0.

Y Is the number of your release of VisualAge Generator — for example, use a **0** (hptcnv40) for VisualAge Generator 4.0.

esf-file-name Is the drive, path, and file name for the external source format file you want to convert.

conversion-table

Is the name of a conversion table that translates from the OS/2 code page to the Windows NT code page.

The converted external source format file is stored as `esf-file-name.cnv` in the directory in which `hptcnvXY.exe` is located.

Use

```
hptcnvXY ?
```

(substituting your version and release numbers for **X** and **Y**) to see a short (English) description of the conversion tool.

5. Use a comparison tool to do a byte-by-byte comparison of the external source format file and the converted file.

If the only character being converted is the not sign, you might be able to use the technique described in "Using the HPTRULES.NLS file" on page 297 to avoid converting all your files. However, use caution if you skip the conversion step — some of your other files might have characters other than the not sign that require conversion.

Be especially alert for any characters being converted that should not be. For example, if you have coded:

```
MOVEA "special-character" TO HIGH-VALUES-CHAR;
```

as a method of setting the high end of a range of keys that you are searching in a file or database, you might not want the special-character to be converted or you might not like the hex value that results from the conversion. If this is the case, you will need to modify your code. A better technique would be to code:

```
MOVEA "special-hex-characters" TO HIGH-VALUES-HEX;
```

This technique would keep the same special-hex-characters in the external source format and converted files.

6. Use the converted external source format file as input to the MSL Migration Assistance Tool or **VAGen Import**.

Notes:

1. hptcnvXY does not support binary GUI tags.
2. Do not use hptcnvXY if you transferred your external source format file to Windows NT in such a way that the code page conversion was performed. For example, if you transferred the file using *ftp* with the *ascii* option, then the code page conversion should have already been done. You need to check that special characters were converted correctly.
3. Refer to the *VisualAge Generator Client/Server Communications Guide* for information about the conversion tables.

Starting the MSL Migration Assistance Tool

You can run the MSL Migration Assistance Tool on either OS/2 or Windows NT.

Action	Description
From the VisualAge Organizer window, select Tools and then VAGen MSL Migration .	The MSL Migration Part List window is displayed.

Building MSL directories

After you have collected your external source format files as described in “Collecting your source code” on page 296 and handled any code page issues as described in “Handling code page changes” on page 297, you are ready to create MSL directories. The **ESF to MSL** push button on the MSL Migration Part List window enables you to build an MSL directory structure from an external source format file.

Note: Be sure that the drive where you plan to create the MSL directory supports long names. For OS/2 this means the drive must be formatted for High Performance File System (HPFS). If this is not done, you might receive a return code 13 when trying to create the MSLs.

Action	Description
From the MSL Migration Part List window, select the ESF to MSL push button.	A window is displayed prompting you for the name of your external source format file.

Action

Select the drive, path, and file name of the external source format file from which you want to build an MSL directory structure.

Enter the drive and path for the directory and select the **OK** push button.

If there were any parts that contained errors in the external source format files, a "List of parts that could not be read" window is displayed followed by the VAGen Parser Messages window.

Description

An Information Required window is displayed prompting you for the name of a directory into which the MSL Migration Assistance Tool can build the MSL.

If the directory you specified exists, a message is displayed prompting you to confirm the directory name.

If the directory does not exist, a message is displayed asking if you want the MSL Migration Assistance Tool to create the directory.

After you have responded **Yes** to either message, an Operation in Progress window is displayed. When the window disappears, the parts (members) from the external source format file have been created in the MSL directory that you specified. All the actions in "Automatic conversions during 4.0 migration" on page 216 have been performed.

The list contains the names of the parts that were not written to the MSL directory. You can write this list to the System Transcript for future reference.

The VAGen Parser Messages window contains both information and error messages. Parts that had no messages or only information messages are already in the MSL directory. You should ignore the information messages. Examples of the information messages include:

HPT.PE.290.i The CICS/OS attribute value was replaced with the MVS CICS attribute value
HPT.PE.17.i The transaction name on the SEGTRAN attribute is not valid
HPT.PE.21.i The FILETYPE VSAMCICS is no longer supported. It was changed to VSAM.

Action	Description
<p>Parts with error messages were not written to the MSL directory. You need to correct these errors by:</p> <ul style="list-style-type: none"> • Correcting the part in Cross System Product or VisualAge Generator and then exporting the external source format file for the part again and using the ESF to MSL push button to add the part to the MSL directory. • Correcting the external source format file and using the ESF to MSL push button to process the external source format file again. 	

Resetting the sandbox from ENVY

If you have previously migrated a subsystem and are now ready to do another subsystem, you might need to reset the MSL Migration Assistance Tool to reflect the parts that are currently in ENVY. This can happen if you migrate one subsystem, work with it for a while in ENVY, and then decide to migrate your remaining subsystems. Particularly if the subsystems to be migrated now share code with the subsystem that was previously migrated, you should reset the MSL Migration Assistance Tool to reflect the current code in ENVY.

Note: You cannot load an application into the sandbox from your ENVY image in the following situations:

- You added a nonvisual part or a Smalltalk class.
- You added parts for VisualAge Generator control information (generation options, linkage table, resource associations, bind control commands, or linkage editor control statements).

However, you can unload these classes so that the application can be reloaded into the sandbox.

Action	Description
<p>From the VisualAge Organizer window, load into your image any applications (such as your common applications) that you need to have in the sandbox when you migrate your new MSLs.</p>	<p>The applications you need should be displayed in the VisualAge Organizer window.</p>

Alternatively, from the Configuration Maps Browser, load the configuration maps that contain the applications you need in your image.

Action	Description
<p>From the MSL Migration Part List window, select the ENVY App Selection push button.</p>	<p>If there are any applications in the sandbox, a message is displayed asking if they can be deleted.</p> <p>If you respond Yes to the message about deleting existing applications or if there were no applications, a Selection Required window is displayed, prompting you to specify the applications already in ENVY that you want to load into the sandbox. Only applications in your image can be loaded into the sandbox.</p> <p>If you respond No, the list of existing applications in the sandbox is not cleared, and your new selections are added to the existing sandbox list. A Selection Required window is displayed, prompting you to specify the applications already in ENVY that you want to load into the sandbox. Only applications in your image can be loaded into the sandbox.</p> <p>If you respond Cancel, the sandbox is not reset from ENVY.</p>
<p>Select one or more applications from the left pane and then select >> to move them to the right pane.</p>	<p>The applications you selected are listed in the right pane.</p>
<p>Select the OK push button.</p>	<p>The applications and the parts they contain are loaded into the sandbox and marked as <i>Committed</i>.</p> <ul style="list-style-type: none"> • There is no analysis of the parts to determine if there were missing (<i>Not Found</i>) parts. • The timestamp of the part in the sandbox is the timestamp from the edition currently loaded in your image, not the timestamp from the original MSL.
<p>You must specify your MSL library selections again (see “Selecting your MSLs” on page 304).</p>	

Action

Description

If you receive a message saying “Unsuccessful in reading ENVY app into tool”, it is because you have added a nonvisual part, a Smalltalk class, or a VisualAge Generator control information part (generation options, linkage table, resource associations, bind control commands, or linkage editor control statements) to the application. The System Transcript window lists the parts that prevent the application from being loaded into the sandbox. Do the following so that you can load the application into the sandbox:

- If the application and the class are already versioned:
 - Open a new edition for the application.
 - Unload the classes that cause a problem.
- If the application and the class are not versioned yet:
 - Version the classes that cause a problem.
 - Unload the classes that cause a problem.

Then reload the application into the sandbox.

After you have reloaded the application into the sandbox, load those classes back into your image now. This ensures that when you version and release classes after committing new or changed applications to ENVY all the classes for this application will be there.

Selecting your MSLs

The MSL Migration Assistance Tool works from an MSL concatenation sequence to move parts into the sandbox. You specify your MSLs and the concatenation sequence as follows:

Action

Description

From the MSL Migration Part List window, select the **MSL Library Selection** push button.

The MSL Library Selection window is displayed.

In the *Basic MSL directory*, enter the drive and directory for a basic MSL that you want to process. For example:

```
f:\ms1s\mym1
```

The directory you specified is added to the *MSL concatenation* area at the bottom of the MSL Library Selection window.

The basic MSL directory that you specify can be an MSL from a previous release of VisualAge Generator or an MSL that you created using the technique described in “Building MSL directories” on page 300.

Select the **Add** push button.

Action	Description
Repeat the previous step until you have all your basic MSLs defined.	The MSL Library Selection window lists all your basic MSLs. The order in which they are listed is the order in which they will be searched for parts.
If you need to change the MSL concatenation sequence, select one MSL in the <i>MSL concatenation</i> area.	The <i>MSL concatenation</i> area changes to reflect the new MSL concatenation sequence.
Then select one of the following to change the concatenation order:	Repeat this step until you are satisfied with the concatenation sequence.
Move Up To move the selected MSL one position higher (earlier) in the concatenation sequence.	
Move Down To move the selected MSL one position lower (later) in the concatenation sequence.	
Remove To delete the selected MSL from the concatenation sequence. The MSL directory is not deleted; only the concatenation sequence is affected.	
When you have the MSL directories listed in proper order for your concatenation sequence, select OK .	The Part List Selection Criteria View window is displayed, with the MSL directories you specified listed under <i>Libraries</i> . The VG Part Prerequisites View window is also displayed.

Selecting and migrating VAGen parts

To migrate a VAGen part and its associates to an ENVY application, you first select the part and then indicate whether you want it to be placed in an existing ENVY application or in a new application. You can work with a group of VAGen parts or a single VAGen part at a time.

This step only moves the parts to the sandbox; they are not moved to the ENVY library manager until you commit the applications to ENVY. It is easier to change the organization of your parts while they are in the sandbox than after they are in the ENVY library manager.

Action	Description
<p>From the Part List Selection Criteria View window:</p> <ul style="list-style-type: none"> • Select the VAGen part type(s) you want to process. Use the All Types push button to select all the part types. Use the Reset push button to deselect all the part types. If you are migrating from MSLs on OS/2, select the Process and/or Statement Group part types. If you are migrating from pseudo MSLs created from external source format files, select the Function part type. • Use a wildcard in the Part name field to limit the search. • Select the Only parts not processed toggle button to limit the list of VAGen parts to those that have not been processed by the MSL Migration Assistance Tool. Deselect the Only parts not processed toggle button to include all the parts that satisfy the part type and wildcard search criteria. • Select the Build List push button. <p>Note: Unlike the VisualAge Generator Member Selection List, all libraries listed in the <i>Libraries</i> area of the Part List Selection Criteria View window are searched for parts. You cannot limit the search to certain MSLs by highlighting the directories listed in the <i>Libraries</i> area.</p>	<p>The MSL Migration Part List window is displayed with the selected VAGen parts.</p> <p>The Status column is set to <i>Processed</i> for any parts that have already been moved to the sandbox.</p>
<p>To help locate parts within the MSL Migration Part List window, you can sort the parts by selecting VAGen Parts and then selecting one of the following:</p> <ul style="list-style-type: none"> • Sort by Type • Sort by Name • Sort by Library 	<p>The order of the parts is changed to match your specified sort criteria.</p>

Action	Description
<p>From the MSL Migration Part List window, select the VAGen part(s) that you want to process with the MSL Migration Assistance Tool. The selected VAGen part and any of its associates that have not yet been migrated will be processed together.</p> <p>To select multiple parts, hold down the Ctrl while you select each of the parts you want to process.</p> <p>You can also select VAGen Parts and then one of the following to help in selecting parts:</p> <ul style="list-style-type: none"> • Find to select one or more parts that satisfy your selection criteria. You can use a wildcard when you specify the selection criteria. For example, if you specify <i>V*</i>, all parts that begin with <i>V</i> are selected. If you specify only a <i>V</i>, the one part named <i>V</i> (if it exists) is selected. • Select All to select all parts in the list. • Deselect All to deselect all parts in the list. • Select Parts Not Processed to select only those parts that are not yet in the sandbox. • Select Not Found Parts to select only those parts that are listed as Not-Found in the Status column. • Select Duplicate Parts to select only those parts with True in the Duplicate column. 	<p>The selected parts are highlighted. If you use Find, Select All, or Select Parts Not Processed any previously selected parts are deselected.</p>
<p>Select VAGen Parts, then Selected and then one of the following:</p> <ul style="list-style-type: none"> • Create Single Application to place all of the selected VAGen parts and their associates into the same new application. An Information Required window is displayed that prompts you for the name of the new application. • Create Multiple Applications to place each of the selected programs and its associates into a separate application. Each application will be named <i>xxxxxApp</i>, where <i>xxxxx</i> is the name of the corresponding program. • Add into Application to place all of the selected VAGen parts and their associates into the same application that already exists in the sandbox. In the Selection Required window that is displayed, select the application into which these parts will be added. 	<p>The VG Part Prerequisites View window is displayed. You can then look at the applications that have been changed and see the parts in them.</p> <p>If any associates of the parts that were just moved to the sandbox were already in the sandbox, one or more <i>ApplicationNodes</i> might be created. An <i>ApplicationNode</i> is created for each part or parts that are shared by two explodable applications.</p>

Creating a new application

If you add VAGen parts with their associates to an ENVY application, and then review the list of VAGen parts that are now in the ENVY application, you might find that there are some parts (for example, common code) that would be better placed in a separate application, but this separate application does not yet exist.

This section describes how to create a new ENVY application using the MSL Migration Assistance Tool.

Action	Description
From the VG Part Prerequisites View window, select Applications and then Create Application .	An Information Required window is displayed, prompting you for the application name.
Type the name of the ENVY application.	The VG Part Prerequisites View window is refreshed and includes the name of the new application. See “Moving a VAGen part between applications” for information about moving VAGen parts from another application into this new application.

Moving a VAGen part between applications

After adding VAGen parts to an application, particularly after adding a program and its associates to an application, you should review the list of VAGen parts for the application in the VG Part Prerequisites View window to determine whether any VAGen parts have been included that would be better placed in a different application. For example, some common code might have been included in the associates list for a program being moved. Reviewing the list of VAGen parts for the application helps to find common code that is better placed in a separate application. If you used naming conventions to distinguish common code, these common VAGen parts are fairly easy to find in the **VAGen Parts** pane of the VG Part Prerequisites View window.

This section describes the steps to move a VAGen part from one application (FromApplication) to another (ToApplication) using the VG Part Prerequisites View window.

Note: After you commit an application to ENVY, you can no longer move its parts to a different application using the MSL Migration Assistance Tool. Refer to the *IBM Smalltalk User's Guide* for information on moving parts between applications in ENVY.

Action	Description
<p>From the VG Part Prerequisites View window:</p> <ul style="list-style-type: none"> • In the Applications pane, select the FromApplication — the application from which you want to move VAGen parts. • In the VAGen Parts pane, select the VAGen parts you want to move. <p>You can select VAGen Parts and then select one of the following to help in selecting parts:</p> <ul style="list-style-type: none"> – Find Parts to select one or more parts that satisfy your selection criteria. You can use a wildcard when you specify the selection criteria. For example, if you specify V*, all parts that begin with V are selected. If you specify only a V, the part named V (if it exists) is selected. When you use Find Parts, any previously selected parts are deselected. – Sort Parts by Type to group parts based on the part type. Any previously selected parts are still selected. – Sort Parts by Name to sort the parts based on the part name. Any previously selected parts are still selected. 	<p>The Selection Required window is displayed, listing the other applications.</p>
<p>Select VAGen Parts and then Move or Move with associates.</p>	<p>The VG Part Prerequisites View window is displayed, showing the FromApplication without the VAGen parts you moved.</p> <p>If you selected Move, only the selected parts were moved.</p> <p>If you selected Move with associates, the selected parts and their associates in the FromApplication are moved to the ToApplication.</p>
<p>You might need to update the prerequisites for the FromApplication to include the ToApplication.</p> <p>See “Checking consistency of applications” on page 321 for information about doing a consistency check to determine which prerequisites need to be changed. You should do a consistency check on the FromApplication and all its dependent applications.</p>	

Controlling the creation of ApplicationNodes

Some parts are used by many programs and views. For example, records, tables, and data items are typically shared by multiple programs and views. Similarly functions (processes and statement groups) might also be shared.

You might need to have programs and views that share common parts in different ENVY applications. A common part can be stored in one of the ENVY applications that uses the part or placed in an ENVY application that contains just common parts.

When you migrate a part, its associates are considered at the same time. An associate that is not yet in the sandbox is placed in the same ENVY application. An associate that is already in the sandbox is treated differently based on the following:

- If the application containing the associate is marked *explodable*, the associate is moved to a new *ApplicationNode*. The original application and the application being created for the part now being moved to the sandbox specify the new *ApplicationNode* as a prerequisite.
- If the application containing the associate is marked *unexplodable*, the associate is not moved to a new *ApplicationNode*. The application for the part now being moved to the sandbox adds the application containing the associate as a prerequisite.

You can change whether an application is *explodable* or *unexplodable* by changing it in the VG Part Prerequisites View window.

Note: After an application is committed to ENVY, it is *unexplodable*.

Action	Description
To change all applications in the sandbox, select Applications and then one of the following:	The VG Part Prerequisites View window is displayed.
• Set All Unexplodable to mark all the applications in the sandbox as <i>unexplodable</i> .	<i>Unexplodable</i> applications are prefixed with an asterisk (*). The * is not part of the application name.
• Set All Explodable to mark all the applications in the sandbox as <i>explodable</i> .	<i>Explodable</i> applications do not have a prefix.

Action	Description
<p>To change a single application in the sandbox, select the application, then select Applications, then Selected and then one of the following:</p> <ul style="list-style-type: none"> • Toggle Explode to change the current setting for the selected application. • Set Unexplodable to mark the selected application as <i>unexplodable</i>. • Set Explodable to mark the selected application as <i>explodable</i>. 	<p>The VG Part Prerequisites View window is displayed.</p> <p><i>Unexplodable</i> applications are prefixed with an asterisk (*). The * is not part of the application name.</p> <p><i>Explodable</i> applications do not have a prefix.</p>

Renaming an application

You might want to rename an application in the following situations:

- All the parts in an *ApplicationNode* belong together and should become an application with a more meaningful name.
- The name you originally chose for the application would be more meaningful if it was changed.
- You typed the application name incorrectly when you created the application.

Action	Description
<p>From the VG Part Prerequisites View window, select the application you want to rename, then select Applications and then Selected → Rename.</p>	<p>An Information Required window is displayed prompting you for the new application name.</p>
<p>Type the new name for the application.</p>	<p>The VG Part Prerequisites View window is displayed and shows the new application name.</p> <p>Applications that specified the renamed application as a prerequisite have been updated to reflect the new name.</p>

Collapsing an application

You might want to merge one application into another application. This might occur if you decide that the two applications will be maintained by the same developer or that they share so many parts that it would be better to combine them. The MSL Migration Assistance Tool calls this process *collapsing*.

Action	Description
From the VG Part Prerequisites View window, select the application you want to collapse, then select Applications and then Selected → Collapse .	An Information Required window is displayed with a combo box listing the applications that exist in the sandbox.
Select the name of the application into which you want to merge and then select the OK push button.	<p>The VG Part Prerequisites View window is displayed. The application that you collapsed no longer appears.</p> <p>If you select the application into which you merged, the parts displayed include the parts from the application you collapsed.</p> <p>Any application that specified the collapsed application as a required application has been updated to reflect the name of the application into which you merged.</p>

Handling Duplicates

You might have duplicate members. This can occur due to:

- Controlled duplicates such as a message table that differs between subsystems as described in “Multiple subsystems with controlled duplicates” on page 269.
- Unintended duplicates as described in “MSLs that contain unintended duplicates” on page 277.
- Duplicates due to using templates and then using a separate MSL for members that were modified due to business logic as described in “MSLs containing code from VisualAge Generator Templates or BW*Wizard” on page 278.

The MSL Migration Assistance Tool helps identify the duplicates and enables you to select the version that you want to migrate. However, it cannot determine for you which level of code is the current version.

Within a single MSL concatenation sequence, to find duplicates easily in the MSL Migration Part List window, select **VAGen Parts** and then **Sort by Name**.

Controlled Duplicates

When you have controlled (intended) duplicates between subsystems, do the following after committing the first subsystem to ENVY, but before starting to migrate the second subsystem.

Action	Description
From the VisualAge Organizer window, unload the applications from the first subsystem from your image.	The unloaded applications should no longer appear in the Applications pane of the VisualAge Organizer window.
From the VG Part Prerequisites View window, delete all the applications for the first subsystem from the sandbox. See “Deleting an application” on page 324 for information on how to do this.	Only the applications created for the common code that is shared by the subsystems should be listed in the VG Part Prerequisites View window.
If you are not able to delete all the applications for the first subsystem from the VG Part Prerequisites View window, select Applications and then Delete All .	All of the applications are deleted from the VG Part Prerequisites View window. Now you can reload the common applications from ENVY as described in “Resetting the sandbox from ENVY” on page 302.

Unintended Duplicates

When you have unintended duplicates, you need to determine which version of the part is the one that should be committed to ENVY. These parts will have *True* in the **Duplicate** column. The options for handling duplicates are:

- Remove Duplicate
- Replace Existing

If you remove a duplicate part:

- The version of the part is removed from future consideration and will no longer appear on the MSL Migration Part List, even if you specify *Only parts not processed*.
- The *removed duplicate* part is ignored for any other processing. For example, if the part is an associate of something moving to the sandbox, the first found version that has not been removed, is what will be moved to the sandbox.

If you replace an existing part in the sandbox:

- The part and any associates that are not currently in the sandbox are moved to the sandbox. The part is moved to the same application where the existing part is already located. The associates are handled as follows:
 - If the associate is already in the sandbox, then
 - If the associate is in the same application as the part, nothing happens.
 - If the associate is in a different application from the part, and is in an unexplodable application, nothing happens.
 - If the associate is in a different application from the part, and is in an explodable application, an ApplicationNode is created.
 - If the associate is not yet in the sandbox, then:

- If the associate is not a duplicate (or you have previously removed duplicates so this is the only version of the associate still in consideration) in the current MSL concatenation sequence, it is moved to the sandbox in the same application as the part that is being replaced.
- If the associate is a duplicate in the current MSL concatenation sequence, the first found version from the concatenation sequence is moved to the sandbox in the same application as the part that is being moved. If you display the MSL Migration Part List window, the duplicates will be listed and you can then Replace Existing if you want something other than the first found associate to go into the sandbox.
- All parts with the same name on the MSL Migration Part List window are updated so their **Status** and **Last Migration Library Timestamp** reflect the part that is now in the sandbox. In addition, any associated parts that are moved to the sandbox are updated on the MSL Migration Part List.

Which options are available for a particular duplicate part depends on the version of the part listed on the MSL Migration Part List and on the version in the sandbox.

- If the **Last Migration Library Timestamp** is filled in on the MSL Migration Part List window, you can:
 - Replace duplicate (put this version of the part into the sandbox, replacing the part that is already there)
 - Remove duplicate (remove this version from future consideration)
- If the **Last Migration Library Timestamp** is blank, what you can do depends on whether this version of the part is from the same MSL as what is in the sandbox:
 - If the **Status** is *Not Found*, you can remove the duplicate, but you cannot replace the existing part. This situation occurs when you have a part that was previously identified as *Not Found*, but you have now found two or more versions of it in your current MSL concatenation sequence. After you have removed duplicate versions of the *Not Found* part, you can use **Add Not Found Part** to update the sandbox (see “Handling missing (not found) parts” on page 318).
 - If this version is from the same MSL, you cannot remove the duplicate or replace the existing part.
 - If this version is from a different MSL, you can remove the duplicate, but you cannot replace the existing part. A blank **Last Migration Library Timestamp** in this situation indicates that this version of the part has the same timestamp as the part in the sandbox and is therefore the same version, but from a different MSL.

Action	Description
<p>To remove a duplicate from further consideration:</p> <ul style="list-style-type: none"> From the MSL Migration Part List window, select the version of the part that you do not want to move to the sandbox. Select VAGen Parts and then Selected → Handle Duplicate Parts → Remove Duplicate. 	<p>The MSL Migration Part List window is updated and this version of the part is no longer included in the list.</p>
<p>To replace an existing part in the sandbox:</p> <ul style="list-style-type: none"> From the MSL Migration Part List window, select the version of the part that you want to use to update the sandbox. Select VAGen Parts and then Selected → Handle Duplicate Parts → Replace Existing. 	<p>The MSL Migration Part List window is updated, and the Status and Last Migration Library Timestamp columns are updated.</p> <p>The part is updated in the sandbox. Any associated parts that were not yet in the sandbox have now been moved to the sandbox.</p> <p>If you replaced a part in an application that has already been committed to ENVY:</p> <ul style="list-style-type: none"> The application name in the VG Part Prerequisites View window is changed to indicate that it has been <i>Modified</i> and that a consistency check must be performed. The need for a consistency check is indicated by a tilde (~) to the right of the application name. See “Checking consistency of applications” on page 321 for information on how to check that all required applications are specified. After you have done the required consistency check, you can commit the <i>Modified</i> applications to ENVY. This creates a new edition of the duplicate parts in ENVY. The part names of the duplicate parts also have a tilde beside them to indicate that they have been modified.

Duplicates for business logic

Duplicates occur for business logic when you have built a program from templates, imported it into one MSL, and then made changes to some members in another MSL. These duplicates appear during migration after you have migrated and committed the template-built parts from their MSL and then change to the MSL that contains the business logic. When you try to migrate the business logic MSL, there will be two types of parts:

- Parts that were not originally built by the templates, but which you added for business logic. These parts will have blanks in the **Duplicate** and **Last Migration Library Timestamp** columns in the MSL Migration Part List window. You handle them like any other parts. In most cases, you will want to add them into an existing (committed) application. The MSL Migration Assistance Tool will mark the application as having been *Modified* and you will be able to commit the application to ENVY again to put the new parts into the ENVY library manager.
- Parts that were originally built for the templates that you modified for business logic. These parts will have *True* in the **Duplicate** column and the timestamp from the template-built MSL in the **Last Migration Library Timestamp** column. You handle these parts as described in the following steps.

Action	Description
Select the duplicate parts. Select VAGen Parts and then Select Duplicate Parts to select all the duplicates at one time.	Each selected part is moved to the sandbox and placed in the same application in which it already exists.
Next select VAGen Parts and then Selected → Handle Duplicate Parts → Replace Existing .	<p>The application name in the VG Part Prerequisites View window is changed to indicate that it has been <i>Modified</i> and that a consistency check must be performed. The need for a consistency check is indicated by a tilde (~) to the right of the application name. See “Checking consistency of applications” on page 321 for information on how to check that all required applications are specified. After you have done the required consistency check you will be able to commit the <i>Modified</i> applications to ENVY. This creates a new edition of the duplicate parts in ENVY</p> <p>The part names of the duplicate parts also have a tilde beside them to indicate that they have been modified.</p>

Finding the application in which a part is located

You might need to determine which application a part or group of parts is currently located in. For example, you might have some common code for which the part names all start with XYZ and you want to verify that you have placed all the parts that begin with XYZ into the same application.

Action	Description
From the VG Part Prerequisites View window, select Applications and then Find Parts .	An Information Required window is displayed.
You can use a wildcard when you specify the selection criteria. For example, if you specify V*, all parts that begin with V are selected. If you specify only a V, the part named V (if it exists) is selected. When you use Find Parts , any previously selected parts are deselected.	If parts satisfying the selection criteria are located in several applications, a Selection Required window is displayed and you can select the application you want to review.
	If parts satisfying the selection criteria are located in only one application, the VG Part Prerequisites View window is displayed with this application highlighted and all the parts within this application that satisfy the selection criteria also highlighted.

Listing missing (not found) parts

When you move parts to the sandbox, their associates generally move with them. However, in some cases the associates might not exist in your MSL concatenation sequence. This can occur for the following reasons:

- Some parts have not yet been developed for a new project.
- Some parts are intentional duplicates that exist in MSLs that will be processed later. For example, see “Multiple subsystems with controlled duplicates” on page 269.
- Some parts have been lost over time or possibly are in MSLs that you have not considered for migration.

You need to resolve these associates that cannot be found — either determine that there is not a problem or locate the missing code. The steps described in the following table enable you to obtain a list of the missing parts. However, you must handle the resolution.

Action	Description
<p>From the VG Part Prerequisites View window, you can list missing parts for all applications in the sandbox, by selecting Applications and then List All Not Found Parts.</p>	<p>A List of not-found parts is displayed. It shows all parts that should have been moved to the sandbox but for which source could not be found in the MSL concatenation sequence. The application name that expected to contain the missing part is also displayed.</p> <p>If all associated parts are found in the sandbox, an Information window is displayed indicating that there are no <i>not found</i> parts.</p>
<p>Select Write To Transcript to put a copy of this list in the System Transcript window.</p>	<p>If you scroll to the bottom of the System Transcript, it displays the list of parts that could not be found.</p> <p>You can save the System Transcript to a file or print it if you need a hardcopy to help in finding the parts.</p>
<p>From the VG Part Prerequisites View window, you can list missing parts for one application in the sandbox, by selecting the application, then select Applications and then Selected → List Not Found Parts.</p>	<p>A List of not-found parts is displayed. It shows all parts within the selected application that should have been moved to the sandbox but for which source could not be found in the MSL concatenation sequence. The application name that expected to contain the missing part is also displayed.</p> <p>If all associated parts are found in the sandbox, an Information window is displayed indicating that there are no <i>not found</i> parts.</p>
<p>Select Write To Transcript to put a copy of this list in the System Transcript window.</p>	<p>If you scroll to the bottom of the System Transcript, it displays the list of parts that could not be found.</p> <p>You can save the System Transcript to a file or print it if you need a hardcopy to help in finding the parts.</p>

Handling missing (not found) parts

When you find the MSL that contains a part that was identified as *Not found*, you need to move the part to the sandbox. Parts in the current MSL that were previously identified as *Not found* are identified in the MSL Migration Part List window by a **Status** of *Not Found*.

Note: If you have now located multiple versions of the previously *Not Found* part, you must first remove duplicates, as described in “Unintended Duplicates” on page 313. This enables the MSL Migration Assistance Tool to know which of the versions you want to move to the sandbox.

Action	Description
<p>From the MSL Migration Part List window, select the part. If there are several <i>Not Found</i> parts, you can select them all at once by selecting VAGen Parts and then selecting Select Not Found Parts.</p>	<p>The part is moved to the application in the sandbox that currently contains the part. The <i>Not Found</i> indicator to the right of the part name in the sandbox is removed.</p>
<p>Next select VAGen Parts and then Selected → Add Not Found Part.</p>	
<p>If the part was moved to the NotFoundApp because you had already committed applications to ENVY, you can</p>	
<ul style="list-style-type: none"> • Move the part to the committed application which expected to have the part. This changes the status of the committed application to <i>Modified</i> and you will need to commit the application to ENVY again. • Move the part to an application that has not yet been committed to ENVY. This situation occurs in the scenario described in “Multiple subsystems with controlled duplicates” on page 269 for the message table part. The message table part is <i>Not found</i> when the COMMON and DBA MSLs are migrated. However, it exists in the SUBSYS1 MSL and needs to be placed into one of the applications being created for SUBSYS1 when you migrate the SUBSYS1 MSL. 	
<p>This situation can also occur in the scenario described in “Separate production MSLs for each developer” on page 272. In this scenario, each subsystem must develop its own version of any previously <i>Not found</i> parts.</p>	

Checking relationships among applications

You might want to determine what parts in an application are referenced by another application or confirm that all prerequisite relationships have been established. You do this using the techniques described in the following sections:

- “Determining which programs are referenced”.
- “Determining the parts that are referenced” on page 320.
- “Checking consistency of applications” on page 321

Determining which programs are referenced

Called programs are not identified as associates of the functions (processes or statement groups) that call them. However, when you are testing, you need to

have any called programs available. Therefore, when you define your configuration maps, you might want to include the applications containing both the called and calling programs in the same configuration map.

Note: Checking called programs can only be done before you commit applications to ENVY.

Action	Description
Select the application for which you want to determine the programs called by any functions (processes or statement groups) it contains.	A list of called programs is displayed.
Select Applications and then Selected → Programs Referenced .	
Select Write To Transcript to put a copy of this list in the System Transcript window.	

Determining the parts that are referenced

In some cases, you might want to determine which parts from one application are referenced by another. For example, if there is only one part that is used in a required application, you might want to move the part to reduce the number of required applications. This function can be done even if one of the applications involved has been committed to ENVY.

Action	Description
Method 1 - Determining which parts in an application are referenced by parts in a particular dependent application.	A list of associated parts in the selected application that are referenced by parts in the dependent application is displayed.
Select the application and then select one of its dependent applications.	
Select Dependent Applications and then Parts Referenced .	The chain of required applications is not considered.
You can save the list of dependent applications by selecting Dependent Applications and then Write To File . The list is copied to the migration log file, mstmig.log .	Scroll to the bottom of the migration log file to see the list of dependent applications. You can also print the migration log file to use this list as a reference.
Method 2 - Determining which parts in a particular required application are referenced by parts in an application.	A list of associated parts in the required application (and its required applications) that are referenced by parts in the selected application is displayed.
Select the application and then select one of its required applications.	
Select Required Applications and then Cross Reference .	The chain of required applications is considered.

Action	Description
You can save the list of referenced parts by selecting Required Applications and then Write To File . The list is copied to the migration log file, mslmig.log .	Scroll to the bottom of the migration log file to see the list of referenced parts. You can also print the migration log file to use this list as a reference.

Checking consistency of applications

When you move parts from one application to another in the sandbox or when you replace a duplicate part in the sandbox, the required applications might not be updated correctly. To check that all the necessary required applications are specified, do the steps described in this section.

Note: In the VG Part Prerequisites View window, if an application name has a tilde (~) to the right of the name, its required applications might not be correct. Until you check consistency for this application, the MSL Migration Assistance Tool will not allow you to commit the application to ENVY.

Action	Description
From the VG Part Prerequisites View window, you can verify that all required applications are specified for an application, by selecting the application, then selecting Applications and then Selected → Check Consistency .	
If the associates of all parts in the selected application can be found in one of the required applications, an Information window is displayed stating <i>No inconsistency found</i> .	Select the OK push button to close the Information window.
If one or more of the associates of some parts in the selected application cannot be found in the required applications, the List of inconsistency window is displayed. It shows the parts that could not be referenced based on the current required applications. The application which needed to use each of the parts is also shown.	See “Finding the application in which a part is located” on page 316 for information about how to find a part in the sandbox.
Select Write To Transcript to put a copy of this list in the System Transcript window.	See “Changing the list of required applications” on page 322 for information about how to change the list of required applications.

Updating the list of required applications

After you have checked the consistency of an application, you might need to modify its list of required applications. In addition, you might want to ensure that the list of required applications does not include any unnecessary applications. These techniques are described in:

- “Changing the list of required applications” on page 322
- “Normalizing the list of required applications” on page 322

Changing the list of required applications

Sometimes you might want to add or delete a required application from the list. For example, after running a consistency check as described in “Checking consistency of applications” on page 321, you might need to add a required application.

Action	Description
From the VG Part Prerequisites View window, select the application for which you want to change its list of required applications. Then select Required Applications and then Change .	A Selection Required window is displayed.
You can: <ul style="list-style-type: none">• Add a required application by moving it from the left pane to the right pane.• Remove a required application by moving it from the right pane to the left pane.	The VG Part Prerequisites View window is refreshed and shows the updated list of required applications.

When you are satisfied with the list of required applications, select the **OK** push button.

Normalizing the list of required applications

You might want to check that unnecessary applications are not included in the list of required applications. However, when you commit the applications to ENVY:

- For GUIs (views), only the applications that are actually required are included in the application’s list of prerequisites.
- For 4GL parts, the application’s list of prerequisites is not set by the MSL Migration Assistance Tool because these prerequisites are not used. However, you might want to record this information for use in building configuration maps.

Action	Description
From the VG Part Prerequisites View window, select the application for which you want to normalize the required applications.	The <i>Required Applications</i> pane is refreshed. Any unnecessary applications have been removed.
Then select Applications , and then Selected → Normalize Prerequisites .	

Deleting an ApplicationNode

An ApplicationNode is created when a VAGen part that is already being used by an existing application is an associate of the VAGen parts being put into another application. The new ApplicationNode reflects the VAGen parts that are common with an existing application and the existing application is not marked as *unexplodable*.

You should review the VAGen parts in the new ApplicationNode and determine where they should be placed — in the existing application, in the application to which you were moving VAGen parts, or possibly in a third application that contains common VAGen parts.

If all parts in the ApplicationNode should be moved to the same application, it is easier to collapse the ApplicationNode as described in “Collapsing an application” on page 311.

However, when parts must be moved to different applications, if you move a part with its associates, all other parts in the ApplicationNode might be associates of the part you moved. This creates a situation in which the ApplicationNode exists, but contains no parts. See “Moving a VAGen part between applications” on page 308 for information about moving a VAGen part to a different application.

This section describes how to delete an application node after all VAGen parts in it have been moved to other nodes and all dependents have been removed.

Action	Description
From the VG Part Prerequisites View window, select the ApplicationNode to be deleted.	The VAGen parts, required applications, and dependent applications for the selected application appear in the VG Part Prerequisites View window.
If there are VAGen parts, see “Moving a VAGen part between applications” on page 308 for information on how to move a VAGen part or “Collapsing an application” on page 311 for information on collapsing the entire ApplicationNode into one application.	
If there are dependents, see “Changing the list of required applications” on page 322 for information on deleting the dependents.	
After all the VAGen parts and dependents have been removed, then select Applications and then Selected → Delete .	The VG Part Prerequisites View window is refreshed and the ApplicationNode has been deleted.

Deleting an application

After you have migrated one subsystem to ENVY, you might need to delete the applications created for that subsystem from the sandbox. This occurs in the scenarios described in “Multiple subsystems with controlled duplicates” on page 269, “Separate production MSLs for each developer” on page 272 and “MSLs that contain unintended duplicates” on page 277.

Deleting one application

To delete one application, follow the steps described below.

Action	Description
From the VG Part Prerequisites View window, select the application to be deleted.	The VAGen parts, required applications and dependent applications for the selected application appear in the VG Part Prerequisites View window.
The application you want to delete must not be specified as a required application by any other application. For example, if AppA lists AppB as a Dependent Application , AppB also lists AppA as a Required Application . You cannot delete AppA until you remove AppA from the list of required applications for AppB.	The list of Dependent applications must be empty.
The applications listed in the Dependent applications pane are the applications that specify the application you want to delete as a required application. If there are dependents, see “Changing the list of required applications” on page 322 for information on deleting this application from the dependents’ list of required applications.	
Select Applications and then Selected → and then Delete .	If the application has no dependents, it is deleted. If it has dependents, an error message is displayed and the application is not deleted.

Deleting all applications

You might want to delete all applications in the sandbox in the following situations:

- After a pilot migration, you might want to clear the sandbox without committing any applications to ENVY so that you can try a different organizational structure for your applications.
- You have finished migrating a group of applications that shared one set of common parts and you want to reset the sandbox before migrating another group of applications with a different set of common parts.

Action	Description
Select Applications and then Delete All .	The VG Part Prerequisites View window is refreshed with all applications cleared from the window.

Committing to ENVY

When you are satisfied with the applications, their VAGen parts, and their lists of required applications, you need to commit this work to the ENVY manager to save it. You can commit one application at a time or all applications at once.

If you commit a single application that contains GUIs and its required applications have not already been committed to ENVY, the MSL Migration Assistance Tool automatically commits the required applications before committing the application you specified. This ensures that the prerequisites for applications that contain views (GUIs) are established correctly.

If an application does not contain GUIs, its required applications are not automatically committed.

Action	Description
<p>To make sure that all parts have been considered, from the Part List Selection Criteria View window, do the following:</p> <ul style="list-style-type: none"> • Select the All Types push button • Select the Only parts not processed toggle button • Select the Build List push button. 	<p>The MSL Migration Part List window is refreshed and contains any parts that have not been moved to the sandbox.</p> <p>Generally, there should not be any parts on this list. If there are, you should determine whether they represent obsolete parts or parts that should be migrated now. For example, you might have parts in your common MSL that are not referenced within the MSL itself but which will be referenced when you migrate MSLs for your subsystems. You should migrate these common parts at this time</p> <p>This helps to insure that all parts are considered by the MSL Migration Assistance Tool.</p>

Action**Description**

From the VG Part Prerequisites View window, you should also do the following:

- Review the number of parts in each application. This number is listed to the right of the application name in the VG Part Prerequisites View window. If the number of parts of any one type is greater than 600-700, you might want to consider splitting this application into smaller applications to improve performance.
 - List any parts not found (see “Listing missing (not found) parts” on page 317).
 - For any application with a tilde (~) to the right of its name, check that the required applications are specified (see “Checking consistency of applications” on page 321).
 - Check which programs are called by an application (see “Determining which programs are referenced” on page 319).
 - Normalize applications to remove unnecessary required applications (see “Normalizing the list of required applications” on page 322).
-

From the VisualAge Organizer window, make sure that the team leader is the current user. This ensures that the team leader will become the manager for all the applications and the owner for all the classes that are created when you commit to ENVY. See “Creating users and setting the current user” on page 294 for information on how to define and change users.

Action

From the VG Part Prerequisites View window, select the applications you want to commit. You can select individual applications or use the following to select applications:

- **Find Applications**
- **Select All**
- **Deselect All**

You can specify:

- An application that has never been committed.
- An application that was previously committed to ENVY but which now indicates that it has been *Modified*. A *Modified* application is one that was committed to ENVY, but which you have added parts to or replaced parts within the application.

If there are relationships among GUIs (views) in different applications, the MSL Migration Assistance Tool will automatically commit:

- Any specified required applications that have not yet been committed to ENVY
- Any specified required applications that were previously committed but which now have been *Modified*.

Note: You cannot commit an application that is already committed into ENVY unless it has been modified.

Select **Applications** and then **Selected** → **Commit Into ENVY**.

If you migrated GUIs, be sure to review the messages in *hptguicv.log*. See “Migrating GUIs” on page 226 for details about the conversion of GUIs to views.

If there were any missing parts listed in NotFoundApp, you can print a list of them by using the technique described in “Listing missing (not found) parts” on page 317.

Description

If you are committing modified applications a warning message is displayed for each application. Select **OK** to commit the modified application to ENVY.

If there were any *Not Found* parts in any of the selected applications or their required applications, a **Warning** message window is displayed. You can select **Cancel** to prevent committing any parts into ENVY if you want to resolve the *Not Found* parts first. You can select **OK** if you want to commit the existing parts to ENVY and resolve the *Not Found* parts later. Select **OK** if you know that the parts are in MSLs that you plan to migrate later (for example, if your scenario matches the one described in “Multiple subsystems with controlled duplicates” on page 269).

After the messages, a progress window is displayed. Committing the applications takes about 2 seconds per VAGen part and somewhat longer for a GUI (view).

When all applications have been committed, the VG Part Prerequisites View window is refreshed and each of the applications has the notation *Committed* beside it.

If there were any missing parts, an application called NotFoundApp was created and contains a list of the missing parts. The NotFoundApp was not committed to ENVY.

Action**Description**

Based on the MSL migration scenario that you are following, you should not delete the applications from the VG Part Prerequisites View window until you are certain you do not need them when you migrate additional MSLs. These committed applications can be used if you migrate additional MSLs to determine where duplicates exist and to help resolve the duplicates.

See “Chapter 31. Completing the ENVY setup on Smalltalk” on page 331 and “Chapter 32. Completing your migration on Smalltalk” on page 339 for information on additional steps you might need to take.

Chapter 30. Using VAGen Import to migrate VAGen 2.x and Cross System Product code to Smalltalk

Follow these steps to migrate VisualAge Generator 2.x and Cross System Product code to VisualAge Generator 4.0:

1. Export your existing code to external source format files. You must use your existing VisualAge Generator 2.x or Cross System Product release to create these .esf files.

Export one external source format file for each ENVY application you want to create. See the product documentation for information about how to create external source format files.

Note: If you are migrating from Cross System Product, also read the book *Migrating Cross System Product Applications to VisualAge Generator (Version 3.1)* for information about other tasks you must perform when migrating from Cross System Product to VisualAge Generator.

If you are changing from the OS/2 to the Windows NT development platform, be sure to review “Changing from OS/2 to Windows NT” on page 298 before you start to migrate.

Note: You should not modify the export files.

2. Start VisualAge Generator 4.0.
3. In VisualAge Generator 4.0, create a VisualAge Smalltalk application using these steps:
 - a. From the VisualAge Organizer window, select **Applications->New**. The New Application window is displayed.
 - b. In the New Application window, enter a name for the application and select **OK**. You will use this new application to receive the contents of one of the external source format files that you want to import into VisualAge Generator 4.0.

Create one ENVY application for each .esf file that you created in step 1. For more information about creating applications, see the *VisualAge Generator User's Guide*.

4. From VisualAge Generator 4.0, from the VisualAge Organizer window, select **VAGen Parts->Parts Browser** to open the VAGen Parts Browser.
5. From the VAGen Parts Browser window, select **Parts->Import/Export->VAGen Import**.

6. In the VAGen Import File Selection window, select one external source format (*.esf) file that you want to import and then select the **Open** button.
7. In the VAGen Import window, check to see if there are any parts listed in the Parts with errors list box. If there are, you need to debug the errors before those parts can be imported with the rest of your application. When there are no parts with errors, proceed to the next step.
8. In the VAGen Import window, specify the name of your application in the Target application field. Move all parts to be imported from the Available parts list to the Selected parts list. Then select the **Import** push button. The selected parts are imported into VisualAge Generator 4.0 in the application you specified.
9. In the VAGen Import window, select the **New File** push button and then repeat steps 6, 7, and 8 for each external source format file you need to import. When you have finished importing your external source format files to VisualAge Generator 4.0, select the **Cancel** push button in the VAGen Import window to return to the VAGen Parts Browser window.
10. Version and release all classes and applications you have imported.
11. Save your VisualAge Generator 4.0 image.
12. See “Chapter 31. Completing the ENVY setup on Smalltalk” on page 331 and “Chapter 32. Completing your migration on Smalltalk” on page 339 for information on additional steps you might need to take.

Chapter 31. Completing the ENVY setup on Smalltalk

The following sections describe specific tasks you might need to perform after running the MSL Migration Assistance Tool. These tasks include:

- “Versioning and releasing a view or a VAGen part class”
- “Versioning an application” on page 332
- “Creating a configuration map” on page 333
- “Adding a required map to a configuration map” on page 334
- “Versioning a configuration map” on page 335
- “Changing the manager of a configuration map” on page 335
- “Assigning ownership of a VAGen part class” on page 336
- “Adding group members” on page 336
- “Changing the ownership of a VAGen part class” on page 336
- “Changing the manager of an application” on page 337

Versioning and releasing a view or a VAGen part class

After you have committed your production level VAGen parts to ENVY, you should version and release the views and VAGen part classes to provide a base line that matches the code that runs in your production system. The steps below describe how to version and release views and VAGen part classes for an application.

Note: You can version and release a view or VAGen part class in two separate steps as described below or in a single step by selecting **Version/Release Owned**. Multiple views and VAGen part classes within a single application can be versioned or released at the same time.

Action	Description
From the VisualAge Organizer window: <ul style="list-style-type: none">• In the Applications pane, select an application.• In the Parts pane, select all the VAGen part classes. These are the classes that start with <i>VAGen</i> (for example, <i>VAGenRecords</i>).	<p>Note: In the Parts pane, VAGen part classes that need to be versioned appear in the format:</p> <pre>>VAGenRecords (06/03/99 10:15:30 AM)</pre>

Action	Description
<p>In the Parts pane, press mouse button 2 and select Version → and then one of the following:</p> <ul style="list-style-type: none"> • Name Each to specify a different version name for each of the classes you are versioning. • One Name to specify the same version name for all of the classes you are versioning. Use this option if you are migrating parts built with VisualAge Generator Templates or BW*Wizard or the business logic for those parts. • Use Defaults to use the ENVY-determined defaults for each of the classes you are versioning. The default might be different for each class. 	<p>The Parts pane in the VisualAge Organizer window is refreshed and shows the version number for each VAGen part class.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. In the Parts pane, VAGen part classes that have been versioned but not released appear in the format: <pre data-bbox="837 447 991 496">>VAGenRecords 1.0</pre> 2. You must be the developer of a class to version the class.
<p>In the Parts pane, press mouse button 2 and select Release → Current Version.</p>	<p>The Parts pane in the VisualAge Organizer window is refreshed and shows the version number for each VAGen part class.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. In the Parts pane, VAGen part classes that have been versioned and released appear in the format: <pre data-bbox="837 852 991 900">VAGenRecords 1.0</pre> 2. You must be the owner of a class to release the class.

Versioning an application

You should version your applications to provide a base line that reflects the level of code that you migrated to ENVY.

Action	Description
<p>From the VisualAge Organizer window, in the Applications pane, select one or more applications to be versioned.</p>	<p>Note: In the Applications pane, applications that need to be versioned appear in the format:</p> <pre data-bbox="801 1324 1083 1373">TrbCommonDataApp (06/03/99 10:15:30 AM)</pre>

Action	Description
<p>In the Applications pane, press mouse button 2 and select Version → and then one of the following:</p> <ul style="list-style-type: none"> • Name Each to specify a different version name for each of the applications you are versioning. • One Name to specify the same version name for all of the applications you are versioning. Use this option if you are migrating parts built with VisualAge Generator Templates or BW*Wizard or the business logic for those parts. • Use Defaults to use the ENVY-determined defaults for each of the applications you are versioning. The default might be different for each application. 	<p>The Applications pane in the VisualAge Organizer window is refreshed and shows the version number for each application.</p> <p>Note:</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. In the Applications pane, applications that have been versioned but not released appear in the format: TrbCommonDataApp 1.0 2. You must be the manager of an application to version the application.

Creating a configuration map

A configuration map allows you to define a group of application editions that should all be loaded together into your image. For example, you might have a configuration map that defines the application versions that are currently in your production system. This section describes how to create a configuration map, release applications into the configuration map, and specify any required maps for the configuration map.

Action	Description
<p>From the VisualAge Organizer window, select Tools and then Configuration Maps.</p>	<p>The Configuration Maps Browser is displayed.</p>
<p>Select Names and then Create.</p>	<p>An Information Required window is displayed prompting you for the name of the configuration map.</p>
<p>Type the name of the configuration map and select the OK push button.</p>	<p>The Configuration Maps Browser is displayed, with the name of the new configuration map included in the list. The new configuration map is highlighted.</p>
<p>In the Description pane in the lower right corner, type a description of the configuration map.</p>	
<p>In the Applications pane, press mouse button 2 and select Add.</p>	<p>A Selection Required window is displayed listing the names of existing applications.</p>

Action	Description
<p>In the Selection Required window:</p> <ul style="list-style-type: none"> • In the Names pane, select an application to add to the configuration map. • In the Editions pane, select the edition of the application. • Select the >> push button to add the application edition to the Selected Editions pane. 	<p>The Configuration Maps Browser is displayed, with the applications listed in the Applications pane.</p> <p>Note: You must be the manager of a configuration map to add applications to the configuration map.</p>
<p>Repeat this process until all the needed applications are included in the Selected Editions pane.</p>	
<p>Select the OK push button.</p>	

Adding a required map to a configuration map

You can define a hierarchy of configuration maps by specifying required maps. For example, in the configuration map for a subsystem, you might want to specify the configuration map for common code as a required map.

Action	Description
<p>From the Configuration Maps Browser, in the Config. Expressions pane, press mouse button 2 and then select Add.</p>	<p>An Information Required window is displayed prompting you to specify an expression.</p> <p>Note: You must be the manager of a configuration map to add a configuration expression.</p>
<p>Select the OK push button to accept the default value of true for the configuration expression.</p>	<p>The Configuration Maps Browser is displayed, with true listed in the Config. Expressions pane.</p> <p>Specifying true means that the configuration maps listed in the Required Maps pane will always be loaded before loading the applications for this configuration map.</p>
<p>From the Required Maps pane, press mouse button 2, and then select Add → As First.</p>	<p>A Selection Required window is displayed prompting you for a configuration map and its edition.</p> <p>Note: You must be the manager of a configuration map to add a required map.</p>
<p>Select the prerequisite map and its edition and then select the OK push button.</p>	<p>The Configuration Maps Browser is displayed, with the configuration map listed in the Required Maps pane.</p>

Versioning a configuration map

To provide a base line for your configuration maps that reflects the level of code that you migrated to ENVY, you should version them.

Action	Description
<p>From the Configuration Maps Browser, select the following:</p> <ul style="list-style-type: none">• In the Names pane, select the name of the configuration map.• In the Editions and Versions pane, select the edition you want to version.• In the Editions and Versions pane, press mouse button 2 and then select Version.	<p>An Information Required window is displayed prompting you to specify a version number.</p> <p>Note: You must be the manager of a configuration map to version the configuration map.</p>
<p>In the Information Required window, select the OK push button to use the default version of 1.0.</p>	<p>The Configuration Maps Browser is displayed with the version showing in the Editions and Versions pane.</p>

Changing the manager of a configuration map

Each configuration map can have a different manager. Use these steps to change the manager of a configuration map.

Action	Description
<p>From the Configuration Maps Browser, in the Editions and Versions pane, press mouse button 2 and then select Change Manager.</p>	<p>A Selection Required window is displayed showing the current group members for the configuration map. The current manager is highlighted.</p> <p>Note: You must be the manager of a configuration map to change the manager of that configuration map.</p>
<p>Select the new manager and select the OK push button.</p>	<p>The Configuration Maps Browser is displayed.</p>

Testing a configuration map

You need to test that the configuration maps you have created will load successfully. To do this, you can unload the applications and then reload them using the configuration maps as described below. Alternatively, test the configuration maps by trying to load them into a clean image.

Action	Description
<p>From the VisualAge Organizer window, select all the applications you created.</p> <p>Then select Applications->Unload.</p>	<p>The VisualAge Organizer window is refreshed and no longer shows the applications you created.</p>

Action	Description
From the Configuration Maps Browser, select a configuration map and then select the current version.	The applications in the configuration map are loaded and now appear in the VisualAge Organizer window.
From the Editions and Versions pane, press mouse button 2 and then select Load With Required Maps (or Load if there are no required maps for this configuration map).	
Check the System Transcript window for any error messages.	

Assigning ownership of a VAGen part class

Each VAGen part class within an application can have a different owner. Although developers can version a VAGen part class, the class owner controls the release of VAGen part classes into the application.

The following sections describe how to add members to the authorized group for an application and how to change the owner of a VAGen part class accessed by that application group.

Adding group members

To change the owner of the class, the new owner must be a member of the group. Group members must be defined for one application at a time.

Action	Description
From the VisualAge Organizer window, select the application for which group members are to be added.	A window showing all defined users and the current group members for the application is displayed. The current manager is indicated by a >.
Select Applications and then Group Members .	Note: You must be the manager of an application to add group members for that application.
From the Users pane, select a user and select the >> push button to move the user to the Group Members pane.	
Repeat this step to add any additional users to the group.	
When all new users have been moved to the Group Members pane, select the OK push button.	The VisualAge Organizer window is displayed.

Changing the ownership of a VAGen part class

The same owner can be assigned for multiple classes within a single application at the same time. During migration, it is easiest to assign the same owner to all the VAGen part classes within an application. The owner of these classes might also (most likely) be the manager of the containing application. After migration, you can use these steps to change the owner of a class.

Action	Description
<p>From the VisualAge Organizer window:</p> <ul style="list-style-type: none"> • In the Applications pane, select the application for which new VAGen part class owners are to be assigned. • In the Parts pane, select the VAGen part classes for which you want to assign the same owner. <p>Select Parts and then Owner → Change Owner.</p>	<p>A Selection Required window is displayed showing the current group members for the application. The current owner is highlighted.</p> <p>Note: You must be the manager of an application or the owner of a class to change the owner of that VAGen part class.</p>
<p>Select the new owner and select the OK push button.</p>	<p>The VisualAge Organizer window is displayed.</p>

Changing the manager of an application

Each application can have a different manager. The manager of an application can create an edition of the application and can version the application.

To change the manager of an application, the new manager must be a member of the group for that application. See “Adding group members” on page 336 for information on adding users to a group.

Action	Description
<p>From the VisualAge Organizer window, in the Applications pane, select the application for which the manager is to be changed.</p> <p>Select Applications and then Manager → Change Manager.</p>	<p>A Selection Required window is displayed showing the current group members for the application. The current manager is highlighted.</p> <p>Note: You must be the manager of an application to change the manager of that application.</p>
<p>Select the new manager and select the OK push button.</p>	<p>The VisualAge Organizer window is displayed.</p>

Chapter 32. Completing your migration on Smalltalk

The following sections describe specific tasks that you might need to do to complete your migration to ENVY. These tasks include:

- “Defining control information”
- “Generating programs and packaging views” on page 340
- “Importing work-in-progress” on page 341
- “Migrating VSAM files” on page 343

In addition, “Chapter 33. Hints and tips on Smalltalk” on page 345 provides information to help you use the VisualAge Organizer and VAGen Parts Browser windows.

Note: ENVY provides a variety of ways of doing most tasks and supports multiple development scenarios. This chapter is intended to provide one way, but not necessarily the only way, of performing tasks related to migrating MSLs to ENVY.

Defining control information

Control information that is needed for test and generation must be stored in ENVY applications. This control information consists of:

- Generation options
- Linkage table
- Resource associations
- Bind control information
- Link edit information

The technique for migrating your existing control information is the same for each of the control files. The only difference is the part type that you specify when you create the part.

Action	Description
Make sure you have an open edition of the application to which you want to add a control information part.	The application shows the date and time stamp when the edition was created.
If you do not have an open edition, from the VisualAge Organizer window, select the application and then select Applications and then New Edition .	
From the VAGen Parts Browser window, select Parts and then New .	The New VAGen Part window is displayed.

Action	Description
Specify the name of the new part.	An editor window is displayed.
<p>Select the Other radio button and then from the drop-down list, select the type that corresponds to the control information you need to add:</p> <ul style="list-style-type: none"> • <i>Generation Options</i> • <i>Linkage Table</i> • <i>Resource Associations</i> • <i>Bind Control</i> • <i>Link Edit</i> 	
<p>From the drop-down combination box, select the application to which you want to add the part. The drop-down list shows applications with open editions and applications that already contain the type of control information you specified.</p>	
Select the OK push button.	
To load the control information from an existing file, select File and then Read From File .	A File Specification window is displayed.
Select the drive, directory, and file name from which you want to load an existing control file. Double-click on the file name or select the Open push button to load the file.	<p>The text from the file you specified appears in the editor window.</p> <p>Note: Be sure to review your options for any that need to be changed. For example, /OPTIONS, /LINKAGE, /RESOURCE, /BIND, and /LINKEDIT specified a directory in VisualAge Generator 2.2. Now they should specify a part name that is stored in ENVY.</p>
If you do not have an existing file, you can use the editor to specify your control information. If you loaded an existing file, you can also use the editor to make changes to the control information.	
Close the editor window and specify Yes when asked if you want to save the changes.	The VAGen Parts Browser is displayed.
Press F5 to refresh the contents of the VAGen Parts Browser window.	

Generating programs and packaging views

After you complete the migration of a group of MSLs (for example, after migrating your production MSLs), you should generate all the programs and package all the views for your target environments. Then test the results in the runtime environments. This helps to ensure that you have migrated the correct version of your code.

Refer to the *VisualAge Generator Generation Guide* for more information on how to generate your programs and package your views.

Note:

1. Any programs for the C++ target environments **must** be regenerated for VisualAge Generator 4.0. There is no coexistence of C++ runtime services between VisualAge Generator 2.2 and 4.0. In addition, it is **strongly recommended** that you regenerate all programs for the COBOL environments and package all views to be sure that you migrated the correct level of code.
2. For CICS OS/2, the default *parmform* option in the linkage table was *COMMDATA*. With VisualAge Generator 4.0, the new option *COMMPTR* is the default. Therefore, if you never specified linkage tables for CICS OS/2, you might need a linkage table now.
3. You must generate the VAGen runtime code before you can package a view that contains a 4GL part. From the VisualAge Organizer window, select the application and then **Applications** and then **Generate**→**VAGen Runtime Code**.

After the VAGen runtime code is generated, you can package the application. From the VisualAge Organizer window, select the application, then **Applications**, and then **Make Executable**.

4. When you package an application, you might receive a Debugger window with a message saying “key not found”. This indicates that a part was used in the view, but does not exist. Scroll down through the stack and find a method that includes at:. Select this line and the name of the missing part should appear on the right side of the window. To remove the message, you need to find the part, load it in your image, and add the application that contains the part to the list of prerequisites for the application you are trying to package. Alternatively, you can remove the part from the view (for example, if the part was a place-holder part in a view built from a template).

Importing work-in-progress

To migrate your work-in-progress MSLs, use **VAGen Import**.

Action	Description
<p>Make sure you have an open edition of the application for which you want to change or add parts.</p> <p>If you do not have an open edition, the application manager can open an edition from the VisualAge Organizer window by selecting the application and then selecting Applications and then New Edition.</p> <p>Note: During migration, you probably want to open an edition of the application because you will be versioning after importing the external source format file from each work-in-progress MSL. After migration, you can use a scratch edition when you import if you will not be adding any classes (no new views or VAGen part classes).</p>	<p>The application shows the date and time stamp when the edition was created.</p>
<p>Make sure you are a group member of all the applications for which you want to change or add parts.</p> <p>If you are not a group member, the application manager can add you to the group by following the steps in “Adding group members” on page 336.</p>	<p>The New Application window is displayed.</p>
<p>From the VisualAge Organizer window, select Applications and then New.</p>	<p>The VisualAge Organizer window is displayed with the new application listed in the Applications pane.</p>
<p>Type the name of the application you want to create.</p> <p>Select the OK push button.</p>	<p>The VAGen Parts Browser window is displayed.</p>
<p>From the VisualAge Organizer window, select VAGen Parts and then Parts Browser.</p>	<p>The VAGen Import File Selection window is displayed, requesting information about the external source format file that you want to import.</p>
<p>From the VAGen Parts Browser window, select Parts and then Import/Export → VAGen Import.</p>	<p>The VAGen Import window is displayed. The Parts with errors pane lists any parts that encountered problems during external source format validation. The Available parts pane lists the parts that successfully passed the external source format validation.</p>

Action	Description
Specify the Destination for duplicate parts and the Target application .	<p>Destination for duplicate parts specifies how duplicate parts (parts that are already in your image) are to be treated. When you import your work-in-progress, you should select the Defined application radio button as the Destination for duplicate parts. This means that any part listed in the Selected Parts pane that already exists in your image is replaced in the application in which it already exists. This preserves the application organization that you created in the sandbox with the MSL Migration Assistance Tool.</p> <p>Target application is the default application into which you want to place the parts. When you select Defined application as the method for handling duplicates, only new parts are imported into the target application.</p>
Select parts from the Available parts pane and move them to the Selected parts pane using the >> push button.	<p>After you select the Import push button, the selected parts are imported and the VAGen Import window is displayed again. The parts you imported are removed from the Selected parts pane and also do not appear in the Available parts pane. This enables you to see what parts from the external source format file have not yet been processed.</p>
When you select the Import push button, only parts in the Selected parts pane are imported. This allows you to import some parts into one target application and another group of parts from the same external source format file into a different target application.	

Migrating VSAM files

If you are migrating from Cross System Product, you might have VSAM files that you need to access during test. Copies of these files must be moved to the workstation for use with the Interactive Test Facility. To migrate these VSAM files, do the following:

- If you use variable-length VSAM files on an MVS host system, see the *VisualAge Generator Installation Guide* for information on how to upload and install a VisualAge Generator utility that is required to prepare these files for download.
- For VM, VSE, and MVS host systems, see *Migrating Cross System Product Applications to VisualAge Generator* (SH23-0244-01) for information on how to REPRO and download your VSAM files.

- For information on how to run the VSAM conversion utility for your downloaded MVS files, from the VAGen Parts Browser window, select **Tools** and then **Data File Conversion**. Then select the **Help** push button.

Converting an RTABLE to a Linkage Table

If you have been using the VisualAge Generator middleware RTABLE for communications routing, the RTABLE entries must be moved from the RTABLE to a linkage table. The following example shows a mapping of RTABLE entries to linkage table entries:

```
RTABLE
app1 - - - - - 1u2 LU2C - 1
app2 - - - - - 1u2 LU2C - 1
app3 - - - - - 1u2 LU2C - 1
app4 - - - - - 1u2 LU2B - 1
$ANY - - - - - 1u2 LU2K - 1

LINKAGE TABLE
:calllink applname=app1 linktype=remote remotecomtype=LU2
serverid=LU2C.
:calllink applname=app2 linktype=remote remotecomtype=LU2
serverid=LU2C.
:calllink applname=app3 linktype=remote remotecomtype=LU2
serverid=LU2C.
:calllink applname=app4 linktype=remote remotecomtype=LU2
serverid=LU2B.
:calllink applname=* linktype=remote remotecomtype=LU2
serverid=LU2K.
```

Chapter 33. Hints and tips on Smalltalk

The following sections provide hints and tips related to the various windows in VisualAge Generator.

System Transcript Window

The following tips apply to the System Transcript window:

- If you are not able to unload an application because there are instances of one or more of its classes, try executing one of the following in the System Transcript window:

```
System abtScrubImage
```

or

```
<defined-class-name> allInstances do: [ :each |  
    each become: nil ]
```

In the second command, you replace **<defined-class-name>** with the name of the class that appeared in the error message when you tried to unload the application.

VisualAge Organizer window

The following tips apply to the VisualAge Organizer window:

- Select **Options** and then **Full Menus** to see all the options on the drop-down menus.
- Select **Options** and then **Preferences** to change your preferences. On the **General** page, consider changing the *Preferred Settings View* to **Properties Table (Recommended)**. Do **not** make this change if some of the tools you use with VisualAge Smalltalk do not yet support properties tables.
- EM-API is the category that is used for the ENVY APIs. These methods are useful for writing tools. To see the methods in this category, select **Tools**, then **Category**, and then type EM-API when you are prompted for the category.

VAGen Parts Browser window

The following tips apply to the VAGen Parts Browser window:

- Under **View** there is an option to **Save as Defaults**. This saves your current *Filter* so that you can bring up the same parts list later. For better

performance in opening the VAGen Parts Browser, you might want to set the *Filter* to only display programs (or no parts at all) and then use **View->Save as Defaults**.

- Under **View**, there is a **Font** setting to set the font used in the VAGen Parts Browser window.
- Under **View**, there is **Reorder Columns** to hide, add, or change the order of columns in the VAGen Parts Browser.
- If you sort by **Description**, the performance of bringing up the part list degrades. This is because each part must be opened to read the description for the sort. If you want to see the description for a particular part, select the part. Its description is displayed in the status area at the bottom of the window.

Refreshing the MSL Migration Assistance Tool

Note: This step is only necessary if there is an updated version (for example, a fix test) sent separately from the VAGen 4.0 product fixpaks.

If an updated version of the MSL Migration Assistance Tool is provided in a separate **.dat** file, it must be imported into the ENVY library and then loaded into your image before you can run it. Make sure the **.dat** file is located on a drive and path that is known to the library management server (EMSRV).

Action	Description
From the VisualAge Organizer window, select Applications , then Import/Export → Import Applications .	An Information Required window is displayed.
In the Information Required window, type the IP name of your library management server (EMSRV) and select OK .	The Enter the full path name of the library window is displayed.
In the Enter the full path name of the library window, select the drive, directory, and file name for the file containing the MSL Migration Assistance Tool (for example, <i>e:\vagen40\dat\migrate.dat</i>) and select Open .	The Selection Required window is displayed.

Action	Description
Under Names , select the first application listed.	A message is displayed indicating that the application is being imported.
Under Versions , select the current version.	
Select the >> push button to move the current version of the application into the Selected Versions list.	
Repeat these steps until all the applications are moved from the left pane to the right pane.	
Make a note of the application names. They are also recorded in the System Transcript window.	
Select the OK push button.	
From the VisualAge Organizer window, select Applications and then View → Show All Applications .	The VisualAge Organizer window is refreshed and now shows all applications.
Select the first application name that was included in the <i>.dat</i> file. Then select Applications and Load → Another Edition .	A list of editions for the application appears.
Select the current version and select the OK push button.	The VisualAge Organizer window is refreshed and now shows the current version of the application.
	You might receive a message about pointers being recached.
Repeat the steps to load all the applications included in the <i>.dat</i> file.	
From the System Transcript window, select File and then Save Image As .	A window asking for the “File name for image?” is displayed.

Action

Select the drive and directory for where you installed VisualAge Generator 4.0 and specify a file name, for example mig1.icx.

Description

Saving a copy of the image as mig1.icx file enables you to start the migration from this point, without having to import and load the MSL Migration Assistance Tool again. To restart from this point at a later time, do the following from an OS/2 window or Windows NT command prompt:

```
/* rename your current image
/* (abt.icx) to a backup name
/* where xxxxx is an identifier
/*          to help you
/*          remember the point
/*          in time from which
/*          the abt file was
/*          renamed
rename abt.icx abtxxxx.icx

/* copy the image that
/* includes the refreshed
/* MSL Migration Assistance Tool to
/* be the new current image
copy mig1.icx abt.icx

/* start VisualAge from the
/* image with the refreshed
/* MSL Migration Assistance Tool
start abt
```

Part 5. Sharing VAGen 4.0 parts between Java and Smalltalk

Chapter 34. Sharing VAGen 4.0 parts between Java and Smalltalk 351
Moving 4GL parts from Java to Smalltalk	351
Moving 4GL parts from Smalltalk to Java	352

Chapter 35. Sharing VAGen Templates 4.0 specifications between Java and Smalltalk 355
Moving VisualAge Generator Templates specifications from Java to Smalltalk	. . . 355
Moving VisualAge Generator Templates specifications from Smalltalk to Java	. . . 357

Chapter 34. Sharing VAGen 4.0 parts between Java and Smalltalk

If you have installed VisualAge Generator 4.0 on both the Java and Smalltalk platforms, you might need to move code between the platforms. For example, you might have common code that needs to be available in both the Java and Smalltalk development environments. The VAGen Import tool enables you to do this.

Note: VisualAge Generator 4.0 only enables you to move non-GUI (4GL) code between the Java and Smalltalk platforms.

Moving 4GL parts from Java to Smalltalk

To migrate VisualAge Generator 4.0 4GL parts from the Java platform to the Smalltalk platform, use these steps:

1. Start VisualAge Generator 4.0 on Java.
2. From the VisualAge for Java Workbench, select **Workspace->Open VAGen Parts Browser** to open the VAGen Parts Browser window.
3. From the VAGen Parts Browser window, select the projects, packages, and parts you want to export, and then select **VAGen Parts->Import/Export->VAGen Export**.
4. In the VAGen Export File Selection window, specify the file you want to contain the output of your export operation, and then select **Open**. Export one external source format file for each ENVY package you want to export to Smalltalk.

Note: You should not modify the export files.

5. Start VisualAge Generator 4.0 on Smalltalk.
6. From the VisualAge Organizer window, create applications that are needed for any new 4GL parts. Open editions of the applications that contain existing parts you want to copy from Java to Smalltalk.
7. From the VisualAge Organizer window, select **VAGen Parts->Parts Browser**. The VAGen Parts Browser window is displayed.
8. From the VAGen Parts Browser window, select **Parts->Import/Export->VAGen Import** to import the external source format files into VisualAge Generator 4.0 on Smalltalk.
9. In the VAGen Import File Selection window, select one external source format (*.esf) file that you want to import and then select **Open**.

10. In the VAGen Import window, check to see if there are any parts listed in the Parts with errors list box. If there are, you need to debug the errors before those parts can be imported with the rest of your application. When there are no parts with errors, proceed to the next step.
11. In the VAGen Import window, specify the name of your application in the Target application field. Move all parts to be imported from the Available parts pane to the Selected parts pane. Then select **Import**. The selected parts are imported into VisualAge Generator 4.0 on Smalltalk in the application you specified.
12. In the VAGen Import window, select the **New File** push button and then repeat steps 9, 10, and 11 for each external source format file you need to import. When you have finished importing your applications to VisualAge Generator 4.0 on Smalltalk, select **Cancel** in the VAGen Import window to return to the VAGen Parts Browser window.
13. Version and release all classes and applications for the parts you have imported.
14. Save your VisualAge Generator 4.0 on Smalltalk image.

Moving 4GL parts from Smalltalk to Java

To migrate VisualAge Generator 4.0 4GL parts from the Smalltalk platform to the Java platform, use these steps:

1. Start VisualAge Generator 4.0 on Smalltalk.
2. From the VisualAge Organizer window, select **VAGen Parts->Parts Browser** to open the VAGen Parts Browser window.
3. From the VAGen Parts Browser window, select the parts you want to export and then select **VAGen Parts->Import/Export->VAGen Export**.
4. In the VAGen Export File Selection window, specify the file you want to contain the output of your export operation, and then select **Open**. Export one external source format file for each ENVY application you want to export to Java.

Note: You should not modify the export files.

5. Start VisualAge Generator 4.0 on Java.
6. From the VisualAge for Java Workbench, create Java projects and packages that are needed for any new 4GL parts. Open editions of projects and packages that contain existing parts you want to copy from Smalltalk to Java.
7. From the VisualAge for Java Workbench, select **Workspace->Open VAGen Parts Browser** to open the VAGen Parts Browser.
8. From the VAGen Parts Browser window, select **Parts->Import/Export->VAGen Import** to import the external source format files into VisualAge Generator 4.0 on Java.

9. In the VAGen Import File Selection window, select one external source format (*.esf) file that you want to import and then select **Open**.
10. In the VAGen Import window, check to see if there are any parts listed in the Parts with errors list box. If there are, you need to debug the errors before those parts can be imported with the rest of the .esf file. When there are no parts with errors, proceed to the next step.
11. In the VAGen Import window, specify the name of your package in the Target package field. Move all parts to be imported from the Available parts pane to the Selected parts pane. Then select **Import**. The selected parts are imported into the package you specified.
12. In the VAGen Import window, select the **New File** push button and then repeat steps 9, 10, and 11 for each external source format file you need to import. When you have finished importing your .esf files to VisualAge Generator 4.0 on Java, select **Cancel** in the VAGen Import window to return to the VAGen Parts Browser window.
13. Version and release all classes, packages and projects for the Smalltalk applications you have imported.
14. Save your VisualAge for Java workspace.

Chapter 35. Sharing VAGen Templates 4.0 specifications between Java and Smalltalk

If you have installed VisualAge Generator Templates 4.0 on both the Java and Smalltalk platforms, you might need to move code between the platforms. The VAGT Import/Export function enables you to do this.

Moving VisualAge Generator Templates specifications from Java to Smalltalk

To migrate VisualAge Generator Templates 4.0 specifications from the Java platform to the Smalltalk platform, use these steps:

1. Start VisualAge Generator 4.0 on Java.
2. From the VisualAge for Java workspace, select **Workspace->Open VAG Templates Browser** to open the VAG Templates Browser window.
3. From the Select Workspace window, select the VAG Templates workspace that you want to export and select the **OK** button.
4. From the VAG Templates Browser window, select **Tools->VAGT Import/Export**.
5. In the VAGT Import/Export window, select an entity from the left pane, select the instances you want to export in the right pane, and then select the **Export** button.
6. In the File Selection window, select or enter the name of the file into which you want to export the VisualAge Generator Templates specifications, and select **Open**.
7. Export one external source format file for each VisualAge Generator Templates entity that you want to export to Smalltalk. Repeat steps 4, 5 and 6 to export VisualAge Generator Templates instances for each entity you want to export.

Notes:

- a. The export of a VisualAge Generator Templates specification x overwrites the old record of this instance x if it exists; otherwise, x is added to the export file.
- b. All you VisualAge Generator Templates specifications can be saved in the same file. This includes the entity definition, generation parameters and workspace. However, if you have several workspaces, you must create a separate file for each workspace and its corresponding generation parameters. For example, suppose you have two workspaces WS1 and WS2 and a Business Object BO1. You should save the BO1 definition in one file, WS1 and its corresponding

BO1 generation parameters in a second file, and WS2 and its corresponding BO1 generation parameters in a third file.

- c. You should not modify the export files.
8. Start VisualAge Generator 4.0 on Smalltalk.
9. Create a VisualAge Smalltalk application by selecting **Applications->New** from the VisualAge Organizer. In the New Application window, enter a name for the application and select **OK**. You will use this new application to receive the contents of one VisualAge Generator Templates workspace and its specifications that you want to import from Java.
Create an application for each VisualAge Generator Templates workspace that you want to import. For more information about creating applications, see the *VisualAge Generator User's Guide*.
10. From the VisualAge Organizer window, select **Tools->VAGT Tools->Open VAG Templates Browser** to open the VAG Templates Browser window.
11. From the Select Workspace window, in the application you created in step 8, create the VisualAge Generator Templates workspace you want to import.
12. From the VAG Templates Browser window, select **Tools->VAGT Import/Export** to import the workspace.
13. From the VAG Templates Import/Export window, select the **Import** button.
14. In the File Selection window, select or enter the name of the file into which you want to import the VisualAge Generator Templates specifications and then select the **Open** button.
15. In the Available Entities window, select the VisualAge Generator Templates specifications you want to import and then select the **OK** button.
16. To import VisualAge Generator Templates specifications corresponding to generation parameters for another workspace, open a new VisualAge Generator Templates workspace. In the VAG Templates Browser window, select **Workspace->Open** and create a VisualAge Generator Templates workspace in a VisualAge Smalltalk application. Then repeat steps 12–15.
17. When you have finished importing your VisualAge Generator Templates specifications, close the VAGT Import/Export window and close the VAG Templates Browser window.
18. Version and release all classes and applications for the VisualAge Generator Templates specifications you have imported.
19. Save your VisualAge Generator 4.0 on Smalltalk image.

Moving VisualAge Generator Templates specifications from Smalltalk to Java

To migrate VisualAge Generator Templates 4.0 specifications from the Smalltalk platform to the Java platform, use these steps:

1. Start VisualAge Generator 4.0 on Smalltalk.
2. From the VisualAge Organizer window, select **Tools->VAGT Tools->Open VAG Templates Browser** to open the VAG Templates Browser window.
3. From the Select Workspace window, select the VAG Templates workspace that you want to export and select the **OK** button.
4. From the VAG Templates Browser window, select **Tools->VAGT Import/Export**.
5. In the VAGT Import/Export window, select an entity from the left pane, select the instances you want to export in the right pane, and then select the **Export** button.
6. In the File Selection window, select or enter the name of the file into which you want to export the VisualAge Generator Templates specifications, and select **Open**.
7. Export one external source format file for each VisualAge Generator Templates entity that you want to export to Smalltalk. Repeat steps 4, 5 and 6 to export VisualAge Generator Templates instances for each entity you want to export.

Notes:

- a. The export of a VisualAge Generator Templates specification x overwrites the old record of this instance x if it exists; otherwise, x is added to the export file.
 - b. All of your VisualAge Generator Templates specifications can be saved in the same file. This includes the entity definition, generation parameters and workspace. However, if you have several workspaces, you must create a separate file for each workspace and its corresponding generation parameters. For example, suppose you have two workspaces WS1 and WS2 and a Business Object BO1. You should save the BO1 definition in one file, WS1 and its corresponding BO1 generation parameters in a second file, and WS2 and its corresponding BO1 generation parameters in a third file.
 - c. You should not modify the export files.
8. Start VisualAge Generator 4.0 on Java.
 9. Create a Java project and package to store the code you want to import from Smalltalk:
 - a. From the VisualAge for Java Workbench window, open the context menu by clicking the right mouse button on an empty part of the Workbench window.

- b. Select **Add->Project** from the context menu.
- c. In the Add Project window, enter a name for the new project and select the **Finish** push button.
- d. From the Workbench window, select the project you just created, and open the context menu.
- e. From the context menu, select **Add->Package**.
- f. In the Add Package window, enter a name for the new package and select the **Finish** push button.

Create one Java project and package for each VisualAge Generator Templates workspace you want to import from Smalltalk. For more information about creating packages and projects, see the *VisualAge Generator User's Guide*.

10. Open the VAG Templates Browser window. From the VisualAge for Java Workbench window, select **Tools->VAGT Tools->Open VAG Templates Browser**.
11. From the Select Workspace window, in the Java package you created in step 10, create the VisualAge Generator Templates workspace you want to import.
12. From the VAG Templates Browser window, select **Tools->VAGT Import/Export** to import the workspace.
13. From the VAG Templates Import/Export window, select the **Import** button.
14. In the File Selection window, select or enter the name of the file into which you want to import the VisualAge Generator Templates specifications and then select the **Open** button.
15. In the Available Entities window, select the VisualAge Generator Templates specifications you want to import and then select the **OK** button.
16. To import VisualAge Generator Templates specifications corresponding to generation parameters for another workspace, open a new VisualAge Generator Templates workspace. In the VAG Templates Browser window, select **Workspace->Open** and create a VisualAge Generator Templates workspace in a Java package. Then repeat steps 13-16.
17. When you have finished importing your VisualAge Generator Templates specifications, close the VAGT Import/Export window and close the VAG Templates Browser window.
18. Version and release all classes, packages and projects for the VisualAge Generator Templates specifications you have imported.
19. Save your VisualAge Generator 4.0 workspace.

Part 6. Appendixes

Appendix A. Name changes for parts and part classes

Table 20 shows the changes made to VAGen part class names during 4.0 migration.

Table 20. Migration of VAGen Part Class Names

Member Type (VisualAge Generator 2.x and Cross System Product)	VAGen Part Class (VisualAge Generator 3.x)	VAGen Part Class (VisualAge Generator 4.0 on Smalltalk)	Java Class (VisualAge Generator 4.0 on Java)
Application	VAGenPrograms	VAGenPrograms	VAGenPrograms
GUI (for VisualAge Generator only)	Not Applicable (GUIs become views)	Not Applicable (GUIs become views)	Not Applicable
Record	VAGenRecords	VAGenRecords	VAGenRecords
Table	VAGenTables	VAGenTables	VAGenTables
Data Item	VAGenDataItems	VAGenDataItems	VAGenDataItems
Map Group	VAGenMapGroups	VAGenMapGroups	VAGenMapGroups
Map	VAGenMaps	VAGenMaps	VAGenMaps
Process	VAGenProcesses	VAGenFunctions	VAGenFunctions
Statement Group	VAGenStatementGroups	VAGenFunctions	VAGenFunctions
PSB (Program Specification Block)	VAGenPSBs	VAGenPSBs	VAGenPSBs

Table 21 shows the changes made to VAGen control information part names during 4.0 migration.

Note: In VisualAge Generator 2.x and Cross System Product, the control information was in files outside of the MSL.

Table 21. Migration of VAGen Control Information Part Names

File (VisualAge Generator 2.x and Cross System Product)	VAGen Part Class (VisualAge Generator 3.x)	VAGen Part Class (VisualAge Generator 4.0 on Smalltalk)	Java Class (VisualAge Generator 4.0 on Java)
Generation Options	VAGen Options	VAGen Options	VAGen Options
Linkage table	VAGenLinkages	VAGenLinkages	VAGenLinkages
Resource Associations	VAGenResources	VAGenResources	VAGenResources
Bind Control Commands	VAGenBindControls	VAGenBindControls	VAGenBindControls

Table 21. Migration of VAGen Control Information Part Names (continued)

File (VisualAge Generator 2.x and Cross System Product)	VAGen Part Class (VisualAge Generator 3.x)	VAGen Part Class (VisualAge Generator 4.0 on Smalltalk)	Java Class (VisualAge Generator 4.0 on Java)
Linkage Editor Control Statements	VAGenLinkEdits	VAGenLinkEdits	VAGenLinkEdits

Table 22 shows the changes made to VAGen GUI composition editor part names during 4.0 migration.

Note: GUIs did not exist for Cross System Product.

Table 22. Migration of GUI Composition Editor Part Names

Member Type (VisualAge Generator 2.x)	VAGen Part (VisualAge Generator 3.x)	VAGen Part (VisualAge Generator 4.0 on Smalltalk)	Java Part (VisualAge Generator 4.0 on Java)
Data	VAGen Data	VAGen 4GL	VAGen 4GL
Logic	VAGen Logic	VAGen 4GL	VAGen 4GL
Container Details	VAGen Container Details	VAGen Container Details	VAGen Container Details
Variable	VAGen Variable	VAGen Variable	VAGen Variable
File Accessor	VAGen File Accessor	VAGen File Accessor	VAGen File Accessor

Table 23 shows the changes made to VAGTemplates part class names and repartition during 4.0 migration.

Table 23. Migration of VAGTemplates part class names and repartition

Entity type	Description type	VAGTemplates part class (VisualAge Generator Templates 3.x)	VAGTemplates part class (VisualAge Generator Templates 4.x on Smalltalk and Java)
Business Object	Definition	VAGTemplates BusinessObjects	VAGTBusinessObjects
	Generation Parameters	VAGTemplates BusinessObjects	VAGTBusinessObject Prms
	Definition Extensions	VAGTemplates BusinessObjects	VAGTExtensions
Data Element	Definition	VAGTemplates DataElements	VAGTDDataElements

Table 23. Migration of VAGTemplates part class names and repartition (continued)

Entity type	Description type	VAGTemplates part class (VisualAge Generator Templates 3.x)	VAGTemplates part class (VisualAge Generator Templates 4.x on Smalltalk and Java)
	Generation Parameters	VAGTemplates DataElements	VAGTDataElement Prms
	Definition Extensions	VAGTemplates DataElements	VAGTExtensions
Interface Unit	Definition	VAGTemplates InterfaceUnits	VAGTInterface Units
	Generation Parameters	VAGTemplates InterfaceUnits	VAGTInterface UnitPrms
	Definition Extensions	VAGTemplates InterfaceUnits	VAGTExtensions
Relational Table	Definition	VAGTemplates RelationalTables	VAGTRelationalTables
	Generation Parameters	VAGTemplates RelationalTables	VAGTRelational TablePrms
	Definition Extensions	VAGTemplates RelationalTables	VAGTExtensions
Value Style	Definition	VAGTemplates ValueStyles	VAGTValueStyles
	Generation Parameters	no parameters	no parameters
	Definition Extensions	VAGTemplates ValueStyles	VAGTExtensions
Workspace	Definition	VAGTemplates Workspaces	VAGTWorkspaces
	Generation Parameters	no parameters	no parameters
	Definition Extensions	VAGTemplates Workspaces	VAGTExtensions

Appendix B. Planning for Migrating Cross System Product MSLs to ENVY

The following sections list information that you should consider when planning your migration from Cross System Product to VisualAge Generator 4.0. The information is organized as a series of questions to help you collect the information that will be needed during the migration process. These sections focus on the migration of Cross System Product applications to VisualAge Generator 4.0. Refer to the appropriate installation manuals for information about the following:

- Hardware prerequisites
- Software prerequisites
- Installing the hardware
- Installing the software
- Customizing the software
- Performance tuning of development environment
- Performance tuning of target environment

See “Appendix C. Planning for Migrating VAGen MSLs to ENVY” on page 381 you are migrating from an earlier version of VisualAge Generator.

Collecting Information about Your Environment

Use the following tables to collect information about your environment:

- Environment Overview (Table 24 on page 366)
- Application Details (Table 25 on page 369)
- Application Management (Table 26 on page 372)
- DB2 Details (Table 27 on page 374)
- DL/I Details (Table 28 on page 374)
- VSAM Details (Table 29 on page 375)
- Special Scheduling Situations (Table 30 on page 375)
- Education Requirements (Table 31 on page 376)
- Integral-Specific (Table 32 on page 376)
- Migration Difficulty Summary - COBOL Generation Issues (Table 34 on page 377)
- Migration Difficulty Summary - General Issues (Table 34 on page 377)

Table 24. Environment Overview Questions

Description	Information
Cross System Product (Version & Releases)?	
Development	_____
Runtime	_____
Note:	
<ol style="list-style-type: none"> 1. If 3.2.1 or earlier, external source format is not available to use in migrating to VisualAge Generator. 2. If 3.2.2 or earlier, CSP/AE (interpretive) was used. 3. If 3.3, determine if COBOL or interpretive was used for the MVS Batch environment; IMS/VS and IMS BMP used generated COBOL; all other environments used interpretive code. 	
What are the current Cross System Product development environments?	
MVS/TSO	_____
MVS CICS (list release)	_____
VSE CICS (list release)	_____
VM	_____
OS/2 (using CSP/2AD)	
What are the current Cross System Product target (runtime) environments?	
MVS Batch	_____
MVS/TSO	_____
IMS/VS or IMS/ESA (list release)	_____
IMS BMP	_____
MVS CICS (list release)	_____
CICS OS/2 (list release)	_____
VSE CICS (list release)	_____
VSE Batch	_____
VM CMS	_____
VM Batch	_____
OS/400	_____
EZ-PREP/EZ-RUN	_____
What is the planned VisualAge Generator development environment?	
OS/2	___ Smalltalk only
Windows NT	___ Smalltalk
	___ Java

Table 24. Environment Overview Questions (continued)

Description	Information
What are the planned VisualAge Generator target (runtime) environments?	
MVS Batch	_____
MVS/TSO	_____
IMS/VS or IMS/ESA (list release)	_____
IMS BMP	_____
MVS CICS (list release)	_____
CICS OS/2 (list release)	_____
CICS/6000 (list release)	_____
VSE CICS (list release)	_____
VSE Batch	_____
VM CMS	_____
VM Batch	_____
OS/2	_____
OS/400	_____
AIX	_____
HP-UX	_____
Solaris	_____
Windows NT CICS	_____
Windows NT	_____
____ GUI Client ____	__ GUI Client __
OS/2	_____
Windows 95	_____
Windows NT	_____
What databases are used?	
DB2 (list release)	_____
SQL/DS (list release)	_____
VSAM	_____
DL/I (list release)	_____
Other (list)	_____
How many application developers use Cross System Product?	_____
Are there any Cross System Product application naming conventions:	
Documented	_____
Followed	_____

Table 24. Environment Overview Questions (continued)

Description	Information
Are there any Cross System Product application development standards: Documented Followed	_____ _____
Can you provide a copy of the documentation?	_____
When was Cross System Product first used in the company?	_____ _____ _____
What Cross System Product features are favorable? Unfavorable?	_____ _____ _____
What are the near-term application development plans?	_____ _____ _____
What are the long-term application development plans?	_____ _____ _____
What is the support structure for applications? (For example, are there separate development and maintenance groups or is each developer responsible for both development and maintenance on a set of applications)	_____ _____ _____
What are you hoping to achieve with this migration?	_____ _____ _____
What kind of expertise do you have in the following areas: COBOL VisualAge C++ PL/I OS/2 Proposed target environment(s) if these are changing	_____ _____ _____ _____ _____

Table 25. Application Details Questions

Description	Information
Application Information	
Number of application systems	_____
Number of application subsystems	_____
Number of applications within each subsystem	_____
Number of applications that are:	
Simple	_____
Medium	_____
Complex	_____
MSL Information	
Number of MSLs shared by developers (exclude individual developer's MSLs)	_____
Are there separate sets of MSLs (for example, one set of MSLs for use by the developers and another parallel set for use by the librarian OR several sets of parallel MSLs, each set matching the code that is at various levels of test or production)	_____
Number of MSLs with common code, records, data items that are shared across subsystems	_____
What concatenation sequences are used for testing with ITF?	_____
What concatenation sequences are used for generation:	_____
For the developers	_____
For system test	_____
For acceptance test	_____
For production	_____
Native COBOL usage	
Do any applications call native COBOL?	_____
Are any applications called by native COBOL?	_____
If so:	
What is the COBOL release?	_____
How many native COBOL programs are used?	_____
How many Cross System Product applications call native COBOL programs?	_____

Table 25. Application Details Questions (continued)

Description	Information
PL/I usage	
Do any applications call PL/I?	_____
Are any applications called by PL/I?	_____
If so:	
What is the PL/I release?	_____
How many PL/I programs are used?	_____
How many Cross System Product applications call PL/I programs?	_____
Other non-Cross System Product languages	
Do any applications call a program written in something other than COBOL or PL/I?	_____
Are any applications called by a program written in the other language?	_____
If so:	
What is the language?	_____
What is the release?	_____
How many non-Cross System Product programs are used?	_____
How many Cross System Product applications call non-Cross System Product programs?	_____
Did you develop applications using Kimberly Clarke's CSP/ADE models?	_____
Do you market any Cross System Product applications to other companies?	_____
Are there any other special application interfaces?	
Printers	_____
CICS Temporary Storage Queue	_____
CICS Transient Data Queue	_____
IMS SPA	_____
Interfaces to non-Cross System Product systems	_____

Table 25. Application Details Questions (continued)

Description	Information
Segmentation	
Are segmented applications used?	_____
Are there any non-segmented applications?	_____
Are there segmented transaction IDs other than XSPS?	_____
Is EZESEGTR used?	_____
Is EZESEGM used?	_____
Coding Practices	
Do you have the same member in more than one MSL?	_____
Any environment-specific code (for example, EZEUSR or CREATX)?	_____
Any long running applications?	_____
Is the Cross System Product message file used?	_____
Are any parameter groups used?	_____
Do any applications have more than 5000 lines of code?	_____
Do any map groups have more than one map, excluding help maps?	_____
Do you use floating maps?	_____
Is the presentation logic in a separate application from the file or database I/O?	_____
Can any main transaction application transfer to any other main transaction application?	_____
What is the usage of local data items?	_____
Are packed and character fields substructured under one another? (If so, might require work to avoid data exceptions when blanks are in the packed fields.)	_____
Can you provide:	
High level application documentation	_____
User documentation	_____
How many of the applications do not have source code?	
Number due to product use (e.g. a licensed product for which you only have object code)	_____
Number due to loss of source code	_____

Table 25. Application Details Questions (continued)

Description	Information
What kind of expertise do you have in the following areas:	
ENVY	_____
VisualAge Smalltalk	_____
Windows NT	_____
OS/2	_____
REXX	_____
Proposed production environment(s) if these are changing	_____

Table 26. Application Management Questions

Description	Information
What library management system do you currently use:	
TeamConnection	_____
SCLM	_____
Other (specify)	_____

Table 26. Application Management Questions (continued)

Description	Information
What is your library management process?	_____
For host environments, consider the following:	_____
Are there separate regions for development, system test, acceptance test, production and so on?	_____ _____
Does each region have a complete load library or is the acceptance test library concatenated in front of production and so on?	_____ _____
How do you move applications from one region to another - by regenerating or by just copying the load module?	_____ _____
For all environments:	_____
Do you have a build administrator or librarian?	_____
Do the developers and build administrator use different versions of:	_____
Generation options	_____
Resource associations	_____
Linkage table (4.1 only)	_____
Special linkage editor command files (3.3 COBOL or 4.1)	_____
Special bind command files (3.3 COBOL or 4.1)	_____ _____
How do you handle any common code? Consider:	_____
Who has authority to change common code?	_____
How frequently does common code change?	_____
How do you determine what to regenerate after changes?	_____ _____
How do you handle emergency fixes?	_____

What is your backup and recovery process?	_____

Table 26. Application Management Questions (continued)

Description	Information
Is there any customization or special initialization set up for a developer?	<hr/> <hr/>
Do you distribute the programs / systems to multiple CPUs or workstations? If so, how?	<hr/> <hr/> <hr/> <hr/> <hr/>

Table 27. DB2 Details Questions

Description	Information
For testing, will the DB2 tables be located on the host or workstation?	<hr/>
Table Information	
Number of tables	<hr/>
For each table:	
Number of columns	<hr/>
Number of rows	<hr/>
Are DB2 tables from multiple databases and/or on multiple platforms required.	<hr/>

Table 28. DL/I Details Questions

Description	Information
For testing, will the DL/I databases be located on the host or workstation? (For a VSE host, the DL/I databases must be on the workstation.)	<hr/>

Table 28. DL/I Details Questions (continued)

Description	Information
Database Information	
Number of databases	_____
For each database:	
Number of segments	_____
Number of fields per segment	_____
Number of occurrences for each segment	_____

Table 29. VSAM Details Questions

Description	Information
For testing, where will the VSAM files be located?	_____
File Information	
Number of files	_____
For each file:	
Number of fields	_____
Number of records in the file	_____
Do any of the files have multiple record types?	_____

Table 30. Special Scheduling Situations Questions

Description	Information
Are there any software changes that must also be made at the same time? For example, are you migrating to a new release of CICS at the same time VisualAge Generator Server for MVS, VSE, and VM is being installed?	_____ _____ _____ _____
Are there any system changes that have to be scheduled around? For example, is there a mainframe CPU change that must not be in conflict with the migration schedule?	_____ _____ _____

Table 30. Special Scheduling Situations Questions (continued)

Description	Information
Are there any critical dates that must be met or avoided? For example, is there a peak period of business activity in which system changes must not be scheduled?	<hr/> <hr/> <hr/> <hr/>
Are there any hardware availability considerations?	<hr/> <hr/> <hr/> <hr/>

Table 31. Education Requirements

Description	Information
Have you made arrangements for the following education?	
Platform (Windows NT or OS/2)	<hr/>
VisualAge Generator	<hr/>
ENVY	<hr/>
VisualAge Smalltalk	<hr/>
VisualAge for Java	<hr/>
LAN Administration	<hr/>
DB2/2 or DB2/NT Administration	<hr/>

Table 32. Integral-Specific Questions

Description	Information
What products / modules do you own and use?	
Position Control	<hr/>
Benefits	<hr/>
Payroll	<hr/>
Pension	<hr/>
Other (specify)	<hr/>
Do you want to migrate all of these modules? If not, then which ones will you migrate?	<hr/>

Table 32. Integral-Specific Questions (continued)

Description	Information
Have you modified, customized, or enhanced any of the Integral modules to be converted? If so, then which ones?	_____
Which types of data does your Integral system use?	
DB2	_____
VSAM	_____

Table 33. Migration Difficulty Summary - COBOL Generation Issues

Description	Points
Are you at Cross System Product 3.2.2 or lower	+1
If you not using generated COBOL today (for example, you are at Cross System Product 3.2.2 or lower or are using Cross System Product 3.3 for interpretive code)	+1
If interpretive, and you have packed or numeric data substructured under character data or vice versa	+2
If interpretive, and you use dynamic SQL	+2
If interpretive, and you use segmented transaction IDs	+2
You have applications created using Kimberly Clarke's CSP/ADE models	+1
You use CICS as the primary development environment	+1
You have limited COBOL development and problem determination experience	+1
You use (or plan to migrate to) COBOL and / or LE	+1
Total number of points	_____

Table 34. Migration Difficulty Summary - General Issues

Description	Points
You have duplicate member names	+2
You have traditionally found numerous Cross System Product APARs, indicating that you exercise the Cross System Product product code differently from other customers	+1

Table 34. Migration Difficulty Summary - General Issues (continued)

Description	Points
You market Cross System Product applications you write to other customers	+1
Applications interface using CALL or DXFR to PL/I or other non-COBOL programs. Or for CICS target environments, the non-Cross System Product programs expect the COMMAREA to contain pointers to the data.	+1
For the CICS target environment, the Cross System Product applications go through a non-Cross System Product program at the CONVERSE.	+1
You use CICS pseudoconversational (Cross System Product segmented) processing	+1
You have limited or unskilled system programmer resources, especially in the Cross System Product target environments	+1
Application developers are unfamiliar with workstation environment for the planned VisualAge Generator Developer environment	+2
You are changing target environments during migration to VisualAge Generator	+3
Number of products (CICS, DB2, IMS) requiring support for multiple release levels	+____
Number of target environments	+____
You have a complex release structure for the systems and subsystems that will be migrated	+3
Release cycle is less than 1 month long	+1
You use home-grown tools for build/promote	+2
You use unsupported tools for build/promote	+3
Total points from COBOL Generation Issues (Table 33 on page 377)	_____
Total number of points	_____

Collecting Information before Migration

Use the following table to collect information that you will need during the migration:

Table 35. Environment Details Questions

Description	Default	Your Value
High level qualifiers		
CSP/AD (V3.3 or earlier)	CSP330_____	_____
CSP/AE (V3.3 or earlier)	CSP330_____	_____
CSP/370RS (V3.3)	CRS110_____	_____
CSP/370AD (V4.1)	CSP410_____	_____
CSP/370RS (V4.1)	CRS210_____	_____
VisualGen Host Services (V1.1)	ELA110_____	_____
VisualAge Generator Server for MVS, VSE, and VM (V1.2)	ELA.V1R2M0__	_____
DB2 information		
High level qualifier	DSN_____	_____
Subsystem ID	DSN_____	_____
Other product high-level qualifiers		
CICS	DFH_____	_____
COBOL	COB2 for VS COBOL II or IGY for COBOL and CEE for LE	_____

Contact List

The following table provides a place to record key contacts in your organization.

Table 36. Contact List

Support Needed	Person	Phone Number
Cross System Product System Administrator		
Cross System Product Build Administrator / Librarian		
PC Coordinator / Expert		
LAN Administrator		
DBA		
RACF authorization		
CICS systems programmer		
IMS systems programmer		
VTAM systems programmer		

Table 36. Contact List (continued)

Support Needed	Person	Phone Number
Installer for VisualAge Generator Developer		
Installer for VisualAge Generator Server for MVS, VSE, and VM		
Installer for VisualGen Host Services for OS/400		
Installer for VisualAge Generator Server for OS/2, AIX, Windows NT, HP-UX, and Solaris		
MVS/TSO Logon Procedures		
Cancelling TSO session		
APAR support (installing)		
Reporting PMRs to Level 1/2		
Access to locked areas		
Access to the building after hours		
How/where to get print outs		

Appendix C. Planning for Migrating VAGen MSLs to ENVY

The following sections list information that you should consider when planning your migration from VisualAge Generator 2.x or earlier to VisualAge Generator 4.0. The information is organized as a series of questions to help you collect the information that will be needed during the migration process. These sections focus on the migration of VisualAge Generator applications to VisualAge Generator 4.0. Refer to the appropriate installation manuals for information about the following:

- Hardware prerequisites
- Software prerequisites
- Installing the hardware
- Installing the software
- Customizing the software
- Performance tuning of development environment
- Performance tuning of target environment

See “Appendix B. Planning for Migrating Cross System Product MSLs to ENVY” on page 365 if you are migrating from Cross System Product.

Collecting Information about Your Environment

Use the following tables to collect information about your environment:

- Environment Overview (Table 37 on page 382)
- Application Details (Table 38 on page 385)
- Application Management (Table 39 on page 388)
- DB2 Details (Table 40 on page 389)
- DL/I Details (Table 41 on page 390)
- VSAM Details (Table 42 on page 390)
- Special Scheduling Situations (Table 43 on page 390)
- Education Requirements (Table 44 on page 391)
- Migration Difficulty Summary - General Issues (Table 45 on page 391)

Table 37. Environment Overview Questions

Description	Information
VisualAge Generator (Version & Fixpaks)?	
Development	_____
Runtime	_____
What is the current VisualAge Generator development environment?	
OS/2	_____
What are the current VisualAge Generator target (runtime) environments?	
MVS Batch	_____
MVS/TSO	_____
IMS/VS or IMS/ESA (list release)	_____
IMS BMP	_____
MVS CICS (list release)	_____
CICS OS/2 (list release)	_____
CICS/6000 (list release)	_____
VSE CICS (list release)	_____
VSE Batch	_____
VM CMS	_____
VM Batch	_____
OS/2	_____
OS/400	_____
AIX	_____
Windows NT CICS	_____
Windows NT	_____
___GUI Client___	___GUI Client_
OS/2	_____
Windows 95	_____
Windows NT	_____
What is the planned VisualAge Generator development environment?	
OS/2	___OS/2 (Smalltalk only)
Windows NT	___Windows NT (Smalltalk)
	___Windows NT (Java)

Table 37. Environment Overview Questions (continued)

Description	Information
What are the planned VisualAge Generator target (runtime) environments?	
MVS Batch	_____
MVS/TSO	_____
IMS/VS or IMS/ESA (list release)	_____
IMS BMP	_____
MVS CICS (list release)	_____
CICS OS/2 (list release)	_____
CICS/6000 (list release)	_____
VSE CICS (list release)	_____
VSE Batch	_____
VM CMS	_____
VM Batch	_____
OS/2	_____
OS/400	_____
AIX	_____
HP-UX	_____
Solaris	_____
Windows NT CICS	_____
Windows NT	_____
____ GUI Client ____	____ GUI Client ____
OS/2	_____
Windows 95	_____
Windows NT	_____
What databases are used?	
DB2 (list release)	_____
SQL/DS (list release)	_____
VSAM	_____
DL/I (list release)	_____
Other (list)	_____
How many application developers use VisualAge Generator?	_____
Are there any VisualAge Generator application naming conventions:	
Documented	_____
Followed	_____

Table 37. Environment Overview Questions (continued)

Description	Information
Are there any VisualAge Generator application development standards:	
Documented	_____
Followed	_____
Can you provide a copy of the documentation?	_____
When was Cross System Product or VisualAge Generator first used in the company?	_____ _____ _____
What VisualAge Generator features are favorable? Unfavorable?	_____ _____ _____
What are the near-term application development plans?	_____ _____ _____
What are the long-term application development plans?	_____ _____ _____
What is the support structure for applications? (For example, are there separate development and maintenance groups or is each developer responsible for both development and maintenance on a set of applications)	_____ _____ _____
What are you hoping to achieve with this migration?	_____ _____ _____
What kind of expertise do you have in the following areas:	
COBOL	_____
VisualAge C++	_____
PL/I	_____
OS/2	_____
Proposed target environment(s) if these are changing	_____

Table 38. Application Details Questions

Description	Information
Application Information	
Number of application systems	_____
Number of application subsystems	_____
Number of applications within each subsystem	_____
Number of applications that are:	
Simple	_____
Medium	_____
Complex	_____
MSL Information	
Number of MSLs shared by developers (exclude individual developer's MSLs)	_____
Are there separate sets of MSLs (for example, one set of MSLs for use by the developers and another parallel set for use by the librarian OR several sets of parallel MSLs, each set matching the code that is at various levels of test or production)	_____
Number of MSLs with common code, records, data items that are shared across subsystems	_____
What concatenation sequences are used for testing with ITF?	_____
What concatenation sequences are used for generation:	_____
For the developers	_____
For system test	_____
For acceptance test	_____
For production	_____
Native COBOL usage	
Do any applications call native COBOL?	_____
Are any applications called by native COBOL?	_____
If so:	
What is the COBOL release?	_____
How many native COBOL programs are used?	_____
How many VisualAge Generator applications call native COBOL programs?	_____

Table 38. Application Details Questions (continued)

Description	Information
PL/I usage	
Do any applications call PL/I?	_____
Are any applications called by PL/I?	_____
If so:	
What is the PL/I release?	_____
How many PL/I programs are used?	_____
How many VisualAge Generator applications call PL/I programs?	_____
Other non-VisualAge Generator languages	
Do any applications call a program written in something other than COBOL or PL/I?	_____
Are any applications called by a program written in the other language?	_____
If so:	
What is the language?	_____
What is the release?	_____
How many non-VisualAge Generator programs are used?	_____
How many VisualAge Generator applications call non-VisualAge Generator programs?	_____
Do you market any VisualAge Generator applications to other companies?	_____
Are there any other special application interfaces?	
Printers	_____
CICS Temporary Storage Queue	_____
CICS Transient Data Queue	_____
IMS SPA	_____
Interfaces to non-VisualAge Generator systems	_____

Table 38. Application Details Questions (continued)

Description	Information
Coding Practices	
Do you have the same member in more than one MSL?	_____
Any environment-specific code (for example, EZEUSR or CREATX)?	_____
Do any map groups have more than one map, excluding help maps?	_____
Do you use floating maps?	_____
Is the presentation logic in a separate application from the file or database I/O?	_____
Can any main transaction application transfer to any other main transaction application?	_____
What is the usage of local data items?	_____
Can you provide:	
High level application documentation	_____
User documentation	_____
How many of the applications do not have source code?	
Number due to product use (e.g., a licensed product for which you only have object code)	_____
Number due to loss of source code	_____
What kind of expertise do you have in the following areas:	
ENVY	_____
VisualAge Smalltalk	_____
Windows NT	_____
OS/2	_____
REXX	_____
Proposed production environment(s) if these are changing	_____

Table 39. Application Management Questions

Description	Information
<p>What library management system do you currently use:</p> <ul style="list-style-type: none"> TeamConnection _____ SCLM _____ Other (specify) _____ 	
<p>What is your library management process?</p>	<p>_____</p>
<p>For host environments, consider the following:</p> <ul style="list-style-type: none"> Are there separate regions for development, system test, acceptance test, production and so on? Does each region have a complete load library or is the acceptance test library concatenated in front of production and so on? How do you move applications from one region to another - by regenerating or by just copying the load module? 	<p>_____</p> <p>_____</p> <p>_____</p> <p>_____</p> <p>_____</p> <p>_____</p> <p>_____</p> <p>_____</p>
<p>For all environments:</p> <ul style="list-style-type: none"> Do you have a build administrator or librarian? Do the developers and build administrator use different versions of: <ul style="list-style-type: none"> Generation options _____ Resource associations _____ Linkage table _____ Special linkage editor command files _____ Special bind command files _____ 	<p>_____</p> <p>_____</p> <p>_____</p> <p>_____</p> <p>_____</p> <p>_____</p> <p>_____</p> <p>_____</p>
<p>How do you handle any common code? Consider:</p> <ul style="list-style-type: none"> Who has authority to change common code? How frequently does common code change? How do you determine what to regenerate after changes? 	<p>_____</p> <p>_____</p> <p>_____</p> <p>_____</p>
<p>How do you handle emergency fixes?</p>	<p>_____</p> <p>_____</p> <p>_____</p> <p>_____</p>

Table 39. Application Management Questions (continued)

Description	Information
What is your backup and recovery process?	<hr/> <hr/> <hr/> <hr/> <hr/>
Is there any customization or special initialization set up for a developer?	<hr/> <hr/>
Do you distribute the programs / systems to multiple CPUs or workstations? If so, how?	<hr/> <hr/> <hr/> <hr/> <hr/>

Table 40. DB2 Details Questions

Description	Information
For testing, will the DB2 tables be located on the host or workstation?	<hr/>
Table Information	
Number of tables	<hr/>
For each table:	
Number of columns	<hr/>
Number of rows	<hr/>
Are DB2 tables from multiple databases and/or on multiple platforms required?	<hr/>

Table 41. DL/I Details Questions

Description	Information
For testing, will the DL/I databases be located on the host or workstation? (For a VSE host, the DL/I databases must be on the workstation.)	_____
Database Information	
Number of databases	_____
For each database:	
Number of segments	_____
Number of fields per segment	_____
Number of occurrences for each segment	_____

Table 42. VSAM Details Questions

Description	Information
For testing, where will the VSAM files be located?	_____
File Information	
Number of files	_____
For each file:	
Number of fields	_____
Number of records in the file	_____
Do any of the files have multiple record types?	_____

Table 43. Special Scheduling Situations Questions

Description	Information
Are there any software changes that must also be made at the same time? For example, are you migrating to a new release of CICS at the same time VisualAge Generator Server for MVS, VSE, and VM is being installed?	_____ _____ _____ _____

Table 43. Special Scheduling Situations Questions (continued)

Description	Information
Are there any system changes that have to be scheduled around? For example, is there a mainframe CPU change that must not be in conflict with the migration schedule?	_____ _____ _____
Are there any critical dates that must be met or avoided? For example, is there a peak period of business activity in which system changes must not be scheduled?	_____ _____ _____
Are there any hardware availability considerations?	_____ _____ _____

Table 44. Education Requirements

Description	Information
Have you made arrangements for the following education?	
VisualAge Generator	_____
ENVY	_____
VisualAge Smalltalk	_____
VisualAge for Java	_____
Optional if changing platforms:	
– Windows NT	_____
– DB2/NT	_____

Table 45. Migration Difficulty Summary - General Issues

Description	Points
You have duplicate member names	+2
You traditionally found numerous VisualAge Generator APARs, indicating that you exercise the VisualAge Generator product code differently from other customers.	+1

Table 45. Migration Difficulty Summary - General Issues (continued)

Description	Points
You market any Cross System Product applications to other customers.	+1
Applications interface using CALL or DXFR to PL/I or other non-COBOL programs. Or for CICS target environments, the non-VisualAge Generator programs expect the COMMAREA to contain pointers to the data.	+1
For the CICS target environment, the VisualAge Generator applications go through a non-VisualAge Generator program at the CONVERSE.	+1
You use CICS pseudoconversational (VisualAge Generator segmented) processing	+1
You have limited or unskilled system programmer resources, especially in the VisualAge Generator target environments	+1
Application developers are unfamiliar with workstation environment for the planned VisualAge Generator Developer environment	+2
You are changing target environments during migration to VisualAge Generator	+3
Number of products (CICS, DB2, IMS) requiring support for multiple release levels	+____
Number of target environments	+____
You have a complex release structure for the systems and subsystems that will be migrated.	+3
Release cycle is less than 1 month long	+1
You use home-grown tools for build/promote.	+2
You use unsupported tools for build/promote.	+3
Total number of points	_____

Collecting Information before Migration

Use the following table to collect information that you will need during the migration:

Table 46. Environment Details Questions

Description	Default	Your Value
High level qualifiers		
VisualGen Host Services (V1.1)	ELA110_____	_____
VisualAge Generator Server for MVS, VSE, and VM (V1.2)	ELA.V1R2M0__	_____

Table 46. Environment Details Questions (continued)

Description	Default	Your Value
DB2 information		
High level qualifier	DSN_____	_____
Subsystem ID	DSN_____	_____
<hr/>		
Other product high-level qualifiers		
CICS	DFH_____	_____
COBOL	COB2 for VS COBOL II or IGY for COBOL and CEE for LE	_____

Contact List

The following table provides a place to record key contacts in your organization.

Table 47. Contact List

Support Needed	Person	Phone Number
VisualAge Generator System Administrator		
VisualAge Generator Build Administrator / Librarian		
PC Coordinator / Expert		
LAN Administrator		
DBA		
RACF authorization		
CICS systems programmer		
IMS systems programmer		
VTAM systems programmer		
Installer for VisualAge Generator Developer		
Installer for VisualAge Generator Server for MVS, VSE, and VM		
Installer for VisualGen Host Services for OS/400		
Installer for VisualAge Generator Server for OS/2, AIX, Windows NT, HP-UX, and Solaris		
MVS/TSO Logon Procedures		

Table 47. Contact List (continued)

Support Needed	Person	Phone Number
Cancelling TSO session		
APAR support (installing)		
Reporting PMRs to Level 1/2		
Access to locked areas		
Access to the building after hours		
How/where to get print outs		

Appendix D. Notes on Cross System Product migrations

This appendix describes updates to the *Migrating Cross System Product Applications to VisualAge Generator* Version 3.1 document (SH23-0244-01). The information in this appendix should be used in conjunction with that book. References in this appendix without page numbers are to chapters and appendixes in *Migrating Cross System Product Applications to VisualAge Generator*.

Running Cross System Product applications with VisualAge Generator Server for workstation platforms

The following considerations apply for running Cross System Product applications on VisualAge Generator Server for workstation platforms:

- VisualAge Generator Server for workstation platforms now includes a client for the Solaris platform.
- Support for migrating applications from host environments supported by Cross System Product is extended to the Solaris platform.
- When migration from Cross System Product interpretive to C++ on VisualAge Generator Server for OS/2, AIX, Windows NT, HP-UX, and Solaris:

- The print destination in VisualAge Generator Server is the resource that is associated to the file name EZEPRINT in the resource association file.
- Three new NLS codes are available:

VisualAge Generator Code	Description
---------------------------------	--------------------

CHT	Traditional Chinese
------------	---------------------

FRA	French
------------	--------

ITA	Italian
------------	---------

- Chapter 8, "CSP/370RS 1.1 to VisualAge Generator Server for MVS, VSE, and VM," has changed substantially. The updated chapter is printed below and replaces the chapter in the *Migrating Cross System Product Applications to VisualAge Generator* document.

Updated Chapter 8, CSP/370RS 1.1 to VisualAge Generator Server for MVS, VSE, and VM

CSP/AD 3.3 required CSP/370RS 1.1 to generate COBOL programs. The following sections list the considerations for migrating CSP/370RS 1.1 using generated COBOL programs to using VisualAge Generator Server for MVS, VSE, and VM.

You can use VisualAge Generator Server for MVS, VSE, and VM to replace CSP/370RS 1.1 as a runtime environment. You can modify any JCL (batch, IMS, CICS, or MVS/TSO) or CLIST that allocated the CSP/370RS 1.1 load library to use the VisualAge Generator Server for MVS, VSE, and VM load library.

Installation considerations

VisualAge Generator Server for MVS, VSE, and VM must be installed in a separate SMP/E zone and have different target libraries from CSP/370RS 1.1. VisualAge Generator Server for MVS, VSE, and VM does not include any of the COBOL generation function that was included in CSP/370RS 1.1, so do not delete CSP/370RS 1.1 from your system until you have migrated both the COBOL generation and runtime services functions to VisualAge Generator.

If you continue to run CSP/370AD and CSP/370RS after you install VisualAge Generator Server for MVS, VSE, and VM 1.2:

1. You must run CSP/370AD and CSP/370RS in a different CICS region or MVS/TSO system from VisualAge Generator Server for MVS, VSE, and VM 1.2.
2. You cannot transfer (CALL or DXFR) between an application generated with CSP/370RS and one generated with VisualAge Generator 4.0. XFER can only be used for applications running in different IMS regions. The transaction code for the application must be set up to run in the IMS region that uses CSP/370RS or VisualAge Generator Server for MVS, VSE, and VM 1.2, based on the product that the application was generated with.

Procedures

The preparation procedures are shipped in VisualAge Generator Server for MVS, VSE, and VM 1.2. The procedure names are not changed from CSP/370RS 1.1 to VisualAge Generator Server for MVS, VSE, and VM.

However, the VisualAge Generator Server for MVS, VSE, and VM procedures do not work with the preparation JCL generated using CSP/370RS 1.1. Therefore, you must:

1. Install and tailor the VisualAge Generator Server for MVS, VSE, and VM 1.2 procedures, replacing the CSP/370RS procedures.
2. Delete any preparation JCL that was created by CSP/370RS COBOL generation.

Upward compatibility — generating applications, tables, and map groups again

Applications, tables, and map groups generated with CSP/370RS 1.1 must be regenerated using with VisualAge Generator 4.0.

Error routines

If EZEFECE is set to 1 and there is an error routine specified, VisualAge Generator Server for MVS, VSE, and VM returns control to the program on OPEN and CLOSE errors. CSP/370RS 1.1 does not.

Defining PSBs

The ELAPCB macro for VisualAge Generator Server for MVS, VSE, and VM supports only the work database. PSBs for applications generated using CSP/370RS 1.1 might also have used a message database.

If you used a message database, you need to generate the PSB again using the ELAPCB macro from VisualAge Generator Server for MVS, VSE, and VM 1.2.

User messages

The message utility is not shipped with VisualAge Generator. User messages are no longer in VSAM files, DL/I databases, or DB2 tables. Instead, they are in VisualAge Generator tables. You will need to convert your messages to a VisualAge Generator message table and then change the text of any messages in the VisualAge Generator message table. See Appendix F, "Using the Message File Conversion Utility," for more information on converting to message tables.

Glossary

This glossary uses the following cross-references:

- Compare to** Indicates a term or terms that have a similar but not identical meaning.
- Contrast with** Indicates a term or terms that have an opposed or substantially different meaning.
- See also** Refers to a term whose meaning bears a relationship to the current term.

A

advance. In VisualAge Generator prior to V3.0, a command that moves members from one MSL of a concatenation (the source) to the next MSL of the concatenation sequence (the target). After a member is moved to the target MSL, it is deleted from the source MSL.

For Cross System Product, the equivalent function is done by the following steps:

- Concatenate the target MSL first (read/write) and the source MSL second (read-only).
- Copy the members from the source MSL to the target MSL.
- Change the MSL concatenation sequence so the source MSL is first (read/write).
- Delete the members from the source MSL.

application. (1) The use to which an information processing system is put; for example, a payroll application or an order-entry application. (2) From Smalltalk, a collection of defined and extended classes that provides a reusable piece of functionality. An application contains and organizes functionally related classes. It also can contain subapplications and

specify prerequisites. (3) In Cross System Product or VisualAge Generator prior to V3.0, a set of definitions that performs a systematic sequence of operations to produce a specific result when generated using the VisualAge Generator, and run using workgroup or host services. For VisualAge Generator V3.0 or later, compare to *program*.

application controls. Policies that determine which users of a library can access specific applications. See also *image and library controls*.

application manager. (1) A team member who is responsible for the overall state of an application. An application manager coordinates the activities of the application's developers and assigns ownership of classes to team members. (2) A browser from which users can create, delete, manage, or configure applications in their image.

associated member. In Cross System Product or VisualAge Generator prior to V3.0, a member that is associated and shares a commonality with other members.

associated part. In VisualAge Generator V3.0 or later, a VAGen part that is associated and shares a commonality with other VAGen parts.

associates. See *associated member* or *associated part*.

attribute. (1) In VisualAge, data that represents a property of a part. For example, a customer part could have a name attribute and an address attribute. Attributes enable a part's public interface to give other parts access to its properties. An attribute can itself be a part, with its own behavior and attributes. (2) In Cross System Product and VisualAge Generator, within an external source format tag, a keyword that variable data is assigned to. For example, part of an external source format tag is in the form

*keyword=*value, where *keyword=* is the attribute and *value* is the variable data assigned to the attribute.

B

behavior. (1) The set of external characteristics that an object exhibits. (2) The abstract class that provides common behavior for Class and Metaclass objects.

browser. A window that supports one or more programming activities, such as creating new classes or methods, modifying existing classes or methods, or viewing library members.

business logic. Code that is specific to the business function that must be accomplished rather than used for infrastructure. Code for business logic might perform a specific payroll calculation or a complex edit function. Code for business logic must be written for the specific program. Code for infrastructure typically provides error handling, message display, transfer of control between programs, and other functions that are standard for many programs. Code for infrastructure can be built from templates.

C

called parameter list. In Cross System Product or VisualAge Generator, the list of maps, records, and data items passed to a called application. The list is defined during application definition for the called application.

class. The specification of an object, including its attributes and behavior. Once defined, a class can be used as a template for the creation of object instances. Class, therefore, can also refer to the collection of objects that share those specifications. A class exists within a hierarchy of classes in which it inherits attributes and behavior from its superclasses, which exist closer to the root of the hierarchy. See also *inheritance*, *polymorphism*, *metaclass*, *defined class*, and *extended class*.

class definition. The definition of a class, containing:

- The class name
- The type of class
- The immediate superclass for the class
- The variables: instance, class, and class instance
- The pool dictionaries the class uses

class developer. A team member who develops and changes classes. The team member who created an edition of a class is that edition's class developer. Contrast with *class owner*.

class extension. An extension to the functionality of a class defined by another application. The extension consists of one or more methods that define the added functionality or behavior. These methods cannot modify the existing behavior of the defined class; they can only add behavior specific to the application that contains the extended class.

class hierarchy. A tree structure that defines the relationships between classes. A class has *subclasses* down the hierarchy from itself and *superclasses* up the hierarchy from itself. The methods and variables of a class are inherited by its subclasses.

class instance variable. Private data that belongs to a class. The defining class and each subclass maintain their own copy of the data. Only the class methods of the class can directly reference the data. Changing the data in one class does not change it for the other classes in the hierarchy. Contrast with *class variable*.

class method. A method that provides behavior for a class. Class methods are usually used to define ways to create instances of the class. Contrast with *instance method*.

class owner. Team member responsible for the integrity of that class in an application edition. The class owner is responsible for releasing class versions. Contrast with *class developer*.

class variable. Data that is shared by the defining class and its subclasses. The instance methods and class methods of the defining class

and its subclasses can directly reference this data. Changing the data in one class changes it for all of the other classes. Contrast with *class instance variable*.

clean image. An image that was saved after the initial installation of VisualAge Smalltalk and which already has the VisualAge Generator feature and any other features loaded into the image. A backup copy of this image should be stored on the LAN. This backup copy can then be copied and used as necessary for testing the load of configuration maps, building a new development image with your application code, and so on.

collapse. During migration, the process of merging one application into another.

component. A functional grouping of classes and related files within a product. See also *system component*.

concatenated MSL. In Cross System Product or VisualAge Generator prior to V3.0, a group of logically connected member specification libraries (MSLs) that are treated as one MSL.

configuration map. A named group of application editions. A configuration map usually represents a product or one of its major parts.

configuration map manager. A team member who maintains the integrity of a configuration map.

containing application. The application to which a class definition belongs. A class can only be defined in one application in the image. Also referred to as the *defining application*.

D

data item. In Cross System Product or VisualAge Generator, a unit of information defined by length, data type, and other characteristics.

debugger. A software tool used to detect, trace, and eliminate errors in computer programs or other software.

defined class. A new class that a containing application adds to the system. It consists of a textual definition (which defines elements such as instance variables) and zero or more methods (which define behaviors). Contrast with *extended class*.

defining application. The application to which a class definition belongs. A class can only be defined in one application in the image. Also referred to as the *containing application*.

dictionary. In Smalltalk, an unordered collection whose elements are accessed by an explicitly assigned external key. See also *pool dictionary*.

duplicate MSL member. In Cross System Product or VisualAge Generator prior to V3.0, a member with the same name in a member specification library (MSL) exists in more than one of the MSLs currently accessed. One might be a copy of the other, or they might be totally different definitions.

E

edit routine. In Cross System Product or VisualAge Generator, a routine specified for special editing of data that an application user types in a map field. It can be a Cross System Product or VisualAge Generator subroutine (EZEC10 or EZEC11), the name of a statement group, or a table used for editing.

edit table. In Cross System Product or VisualAge Generator, a table named in place of an edit routine in the definition of a map variable field edit. See also *edit routine*.

edition. In VisualAge, a software component that is subject to further change. A software component can have one or more editions, identified by a timestamp stating the date and time of the edition's creation. Many changes can be made to a single edition of a class. In contrast, every change to a method creates a new edition of that method. In its broadest sense, *edition* can include *scratch edition* and *version*.

encapsulation. The hiding of a software object's internal data representation. The object provides

an interface that queries and manipulates the data without exposing its underlying structure.

error routine. In Cross System Product or VisualAge Generator, a routine called when an error occurs while running a process option that accesses a record. The error routine can be a valid EZE word, the name of a process, or the name of a statement group. If an error routine is not specified, the application or program ends when an error occurs, displaying a message describing the error condition.

explodable. For the MSL Migration Assistance Tool, marking an application as *explodable* means that the parts within the application will be automatically moved to a new ApplicationNode if the parts are associates of parts being migrated to a different application. Contrast with *unexplodable*.

export. (1) In Cross System Product or VisualAge Generator prior to V3.0, a command that copies one or more members of a member specification library (MSL) to a serial file. For VisualAge Generator, only external source format files could be exported. Contrast with *import*. For VisualAge Generator V3.0 or later, compare to *VAGen export*. (2) From Smalltalk, a command for moving application versions from your team's current library to a different library. Export works with binary files. Contrast with *File Out*.

ESF. See *external source format*.

extended class. A class that uses and extends the functionality of a class defined by another application. It consists of one or more methods that define the added functionality or behavior. These methods cannot modify the existing behavior of the defined class; they can only add behavior specific to the application that contains the extended class. Contrast with *defined class*.

external source format (ESF). In Cross System Product or VisualAge Generator, a data format that makes it possible for Cross System Product or VisualAge Generator to import member definitions developed outside the product. The format consists of a readable set of tags and attributes. The external source format is also

used to export member specification library (MSL) members for hard-copy printing, editing, and analysis.

F

File In. A Smalltalk command for compiling external definitions of applications, classes, and methods from a text file.

File Out. A Smalltalk command for writing definitions of applications, classes, and methods to an external text file.

first-found member. In Cross System Product or VisualAge Generator prior to V3.0, the member definition that is located first when searching for a specifically-named member in a concatenated member specification library (MSL).

flow stage. In Cross System Product or VisualAge Generator, an optional form of the connecting logic between processes in an application or program. The flow stage contains instructions that control the sequence of the main processes in an application. If flow stage statements are not defined, the next main process in the list is performed by default.

G

generate. In Cross System Product or VisualAge Generator, a command that produces an application, (or program), map group, or table suitable to prepare and run using workgroup services or host services. For VisualAge Generator prior to V3.0, GUI applications could also be generated. Contrast with *package*.

generation options default file. In Cross System Product or VisualAge Generator, a file that contains values for generation options. The name of this file is specified on the EZEROPT environment variable. This is the only options file in which NOOVERRIDE can be specified for an option.

generation options file. In Cross System Product or VisualAge Generator, an input to the

generation process. A file that contains values for generation options in addition to the generation option defaults file.

global data item. (1) In Cross System Product or VisualAge Generator prior to V3.0, a data item that is saved in the member specification library (MSL) and can be referenced by any other member in addition to the record or table for which it was created. Changes to global data items affect all records and tables that include that item as a global item. Contrast with *local data item*. (2) In VisualAge Generator V3.0 or later, a data item that is saved as a VAGen part and can be referenced by any other VAGen part in addition to the record or table for which it was created. Changes to global data items affect all records and tables that include that item as a global item. Contrast with *local data item*.

global variable. A variable that any method in any object can access.

graphical user interface (GUI). A type of interface that enables users to communicate with a program by manipulating graphical elements, rather than by typing commands. Typically, a graphical user interface includes a combination of graphics, pointing devices, menu bars, overlapping windows, and icons.

group member. A team member who belongs to a group that is responsible for developing an application.

GUI. See *graphical user interface*.

GUI application. In VisualAge Generator prior to V3.0, a definition of a window or set of windows defining the graphical user interface for an application, including connections to VisualAge Generator data and logic, that performs operations to produce a specific result. For VisualAge Generator V3.0 or later, compare to *view* and *visual part*.

icon. A small pictorial representation of an object.

image. (1) A Smalltalk file that provides a development environment on an individual workstation. An image contains object instances, classes, and methods. It must be loaded into the Smalltalk virtual machine in order to run. (2) In the Distributed feature, the file in which an object space has been saved; a static collection of objects along with the environment in which they execute.

image and library controls. Access controls that protect the integrity of major system components. For example, such controls can bind the ownership of an image to a specific user with password protection. See also *application controls*.

import. (1) In Cross System Product or VisualAge Generator prior to V3.0, a command that copies one or more members from a serial file to a member specification library (MSL). For VisualAge Generator, only external source format files could be imported. Contrast with *export*. For VisualAge Generator V3.0 or later, compare to *VAGen import*. (2) From Smalltalk, a command for loading application or subapplication versions from a library other than the one your team is currently using. Import works with binary files. Contrast with *File In*.

inheritance. A relationship among classes in which one class shares the structure and behavior of another. A *subclass* inherits from a *superclass*.

Inspector. A Smalltalk tool for viewing the data of any Smalltalk object.

instance. An object that is a single occurrence of a particular class. An instance exists in memory or external media in persistent form.

instance method. In Smalltalk, a method that provides behavior for particular instances of a class. Messages that invoke instance methods are sent to particular instances, rather than to the class as a whole. Contrast with *class method*.

instance variable. Private data that belongs to an instance of a class and is hidden from direct access by all other objects. Instance variables can

only be accessed by the instance methods of the defining class and its subclasses.

L

LAN. Local area network.

level. In Cross System Product or VisualAge Generator prior to V3.0, the degree of subdivision of a data structure. The level numbers subdivide a record or table. In general, higher level numbers indicate subdivisions of preceding lower level numbers. See also *level-77*.

level-77. In Cross System Product or VisualAge Generator, a level that can be defined for unstructured or single data items in working storage. See also *level*.

library. A shared repository represented by a single file. It stores source code, object (compiled) code, and persistent objects, including editions, versions, and releases of software components. See also *system component*.

library supervisor. A user ID that has the capability to add new users to the system.

lineup. A configuration of prerequisites and subapplications that an application requires in order to run on a platform. Boolean expressions specify the platform or conditions under which the applications in the lineup can load.

linkage table. For Cross System Product or VisualAge Generator, an input to the generation process. A file that defines the environment-dependent linkage to be generated for the following:

- transfers to applications or programs
- called applications or programs receiving parameters
- starting asynchronous transactions using the CREATX service routine
- file operations to specific files

list of associates. See *associated member* and *associated part*.

load. A system operation that links the compiled code for a software component from a

library into an active image. Loading also performs other operations that enable the component to run, such as linking prerequisites. Contrast with *unload*.

loaded. Designation for a software component that has been linked from a library into an image.

local area network (LAN). A computer network located in a user's establishment within a limited geographical area. A LAN typically consists of one or more server machines providing services to a number of client workstations.

local data item. In Cross System Product or VisualAge Generator, a data item that is stored with the record or table for which it was defined. Changes to local data items have no effect on definitions of items with the same name in other records or tables. Contrast with *global data item*.

M

map. In Cross System Product or VisualAge Generator, a definition of the format of all or part of what will appear on a screen or printer page. A map is used to define the layout and characteristics of information presented on a display or printer while running an application or program.

map group. In Cross System Product or VisualAge Generator, contains all the maps used by an application or program. Maps from the same map group can be used in several applications or programs.

member. In Cross System Product or VisualAge Generator prior to V3.0, part of a member specification library (MSL), consisting of a name and a complete definition. A member can be a data item, record, table, map, map group, application, process, statement group, PSB, or GUI application. For VisualAge Generator V3.0 or later, compare to *VAGen part*, *view*, and *visual part*.

member specification library (MSL). In Cross System Product or VisualAge Generator prior to V3.0, a single library containing information that

contains the definitions, such as data items, records, maps, and processes, that are used to generate an application. In Cross System Product, an MSL is a VSAM file. In VisualAge Generator, an MSL is a directory. See also *read-only MSL* and *read/write MSL*.

member type. In Cross System Product or VisualAge Generator prior to V3.0, the type of code a member contains. The member types are: data item, record, table, map, map group, application, process, statement group, PSB, and GUI application. For VisualAge Generator V3.0 or later, compare to *VAGen part class*.

metaclass. The specification of a class; the complete description of a class's attributes, behavior, and implementation. Every class has a metaclass, of which it is the sole instance. Contrast with *class*.

method. Executable code that implements the logic of a particular message for a class. In VisualAge, methods are also called *scripts*. See also *class method* and *instance method*.

MSL. See *member specification library*.

MSL member. See *member*.

N

not found part. For the MSL Migration Assistance Tool, a part that is associated with another part, but which could not be found in the MSL concatenation sequence.

nonvisual class. A class in a VisualAge application that specifies a nonvisual part. For example, *Person*, *Address*, and *BankAccount* are nonvisual classes. Contrast with *view*.

nonvisual part. A part that has no visual representation at run time. A nonvisual part typically represents some real-world object that exists in the business environment. Contrast with *view* and *visual part*.

O

object. The basic building block in Smalltalk development. An object is anything that exhibits behavior. All code and data in Smalltalk must be part of an object.

owning application. The application to which a method in a class belongs. The owning application does not have to be the same application as the defining application. The owning application does not have to be the same for all methods of a class.

P

package. (1) In Smalltalk, the process of making a visual part executable. Packaging must be done on the platform in which the visual part will run. (2) In Java, a group of classes and methods that are closely related in function. A package can include VAGen part classes and VAGen parts. Java packages are stored in projects. See also *project*.

pane. A section of a window that shows a particular type of data. For example, the **Applications** pane of the **VisualAge Organizer** window shows a list of ENVY applications. See also *window*.

part. A self-contained software object with a standardized public interface, consisting of a set of external features that enable the part to interact with other parts. The parts on the VisualAge parts palette can be used as templates to create instances of objects.

polymorphism. The ability of different objects to respond to the same message in different ways. This means that different objects can have very different method implementations for the same message. An object can send a message without concern for its underlying implementation

pool dictionary. A dictionary object whose keys define variables that can be shared by multiple classes. All methods for a class can access the

variables in a pool dictionary if the class declares the pool dictionary as part of its scope.

prerequisite. In Smalltalk, prerequisite usually refers to a *prerequisite application*.

prerequisite application. A particular version or edition of an application required by another application for it to function successfully. An application can extend or reference one or more of the prerequisite application's classes.

prerequisite class. A class in a prerequisite application. An application can subclass, extend, and send messages to a prerequisite class.

prerequisite relationship. A relationship that specifies that a particular component must exist before a second component can exist.

process. (1) In Cross System Product or VisualAge Generator, a unit of an application or program. A process is divided into a process option and processing statements. Each process does one task. Main processes can be connected by branching instructions in the flow stage of the application or program. Processes provide a modular design approach. See also *process object*, *process option*, and *processing statement*. Contrast with *statement group*. (2) In Smalltalk, a sequence of actions described by expressions and performed by the system's virtual machine.

processing statement. In Cross System Product or VisualAge Generator, a statement in a process, statement group, or flow definition for a main process. When the application or program runs, processing statements perform the tasks that are needed. Processing statements can be grouped in the following categories: arithmetic statements, data manipulation statements, external branching statements (unconditional), and internal branching statements (conditional and unconditional). See also *process* and *statement group*.

process object. In Cross System Product or VisualAge Generator, the name of a record or map accessed by the process option. See also *process* and *process option*.

process option. In Cross System Product or VisualAge Generator, the function to be performed by the process. Only one process option is permitted in a process. Process options except EXECUTE represent I/O operations. The EXECUTE option is the default process option.

The process options are: ADD, CLOSE, CONVERSE, DELETE, DISPLAY, EXECUTE, INQUIRY, REPLACE, SCAN, SCANBACK, SETINQ, SETUPD, SQLEXEC, and UPDATE. See also *process* and *process option*.

program. In VisualAge Generator V3.0 or later, a set of definitions that performs a systematic sequence of operations to produce a specific result when generated using the VisualAge Generator, and run using workgroup or host services. Contrast with *application*.

program specification block (PSB). A formal DL/I description of the hierarchical data base structures that an application can access. The PSB shows the hierarchical relationship that exists between types of segments. In the IMS environment, program communication blocks (PCBs) for the IMS message queue or GSAM databases are also included. For Cross System Product or VisualAge Generator, the PSB is a subset of the information stored in the DL/I PSB.

project. A group of package editions that should be loaded together into a developer's workspace. See also *package*.

project list part (PLP). A VisualAge Generator generation options part. It is used to identify projects to load before the containing project is loaded. The PLP is the VisualAge Generator on Java way of implementing VisualAge Generator on Smalltalk required maps. See also *project*.

PSB. See *program specification block*.

R

read-only MSL. In Cross System Product or VisualAge Generator prior to V3.0, a type of access to data that permits it to be read but not modified. All MSLs except the first one in the

concatenation sequence are read-only MSLs. See also *member specification library*. Contrast with *read/write MSL*.

read/write MSL. In Cross System Product or VisualAge Generator prior to V3.0, a type of access to data that permits it to be read and modified. Only the first MSL in an MSL concatenation is a read/write MSL. The other MSLs in the concatenation sequence are read-only MSLs. See also *member specification library*. Contrast with *read-only MSL*.

record. In Cross System Product or VisualAge Generator, a collection of related data items treated as one unit. Records are the information units that form a file or database.

release. A system operation on a component that changes its containing component's configuration. Releasing a component adds its released edition or version to the configuration for its containing component. When a containing component is loaded into an image, the released editions or versions of the components it contains are also loaded.

repository. (1) An organized, shared body of information that can support business and data-processing activities. (2) In VisualAge, the multi-user library that stores components such as applications, classes, and methods created by application developers. It stores source code, object code, and persistent objects.

resource association file. In VisualAge Generator, an input to generation. A file that defines how a serial, relative, or indexed file, or printer map is to be implemented for a specific target environment. The default values for system resource names, file type, and program communication block (PCB) numbers can be changed in the resource association file. For Cross System Product, compare to *resource association information*.

resource association information. In Cross System Product, an input to generation that defines how a serial, relative, or indexed file, or printer map is implemented for a specific target environment. The default values for system

resource names, file type, and program communication block (PCB) numbers can be changed by specifying different resource association information. Resource association information is stored in the MSL. For VisualAge Generator, compare to *resource association file*.

R-O. See *read-only MSL*.

R/W. See *read/write MSL*.

S

sandbox. For the MSL Migration Assistance Tool, the applications and parts displayed in the **VG Part Prerequisites View** window in which you can experiment with your organizational structure without committing the parts to ENVY.

scratch edition. A modifiable and private copy of an application for a user who is not necessarily the application's manager. The scratch edition only exists in that user's image. Using a scratch edition, you can modify an application version, and the existing classes contained in it, without actually creating a new edition. Each scratch edition has <<>> displayed around the edition timestamp or version name. Contrast with *edition* and *version*.

script. A series of Smalltalk statements that implement an action for a part. Scripts are equivalent to Smalltalk methods.

Smalltalk (ST). (1) A complete programming environment for developing object-oriented applications. Smalltalk is a pure implementation of object-oriented concepts; every entity in the environment is an object. (2) The name of the programming language that the Smalltalk programming environment supports. (3) In VisualAge, the name of the System Dictionary containing the global variables.

statement group. In Cross System Product or VisualAge Generator, a set of processing statements that perform processing only. No I/O operations are permitted in a statement group. When a statement group finishes running, control is returned to the processing statement

that started running the statement group. See also *processing statement*. Contrast with *process*.

status area. A part of a window where information appears that shows the state of an object or the state of a particular view of an object.

subapplication. An application contained by another application. Using subapplications, you can organize the classes of an application into a tree of subapplications, or isolate the parts of an application that are platform-specific.

subclass. A class that inherits behaviors and specifications (in other words, methods and variables) from another class. Contrast with *superclass*.

superclass. A class from which another class inherits behaviors and specifications (in other words, methods and variables). Contrast with *subclass*.

system component. A component that manages storage of code and access to that stored code. A library file is a system component that stores and manages code. A user object is a system component that represents a person who can use the library. See also *component*.

T

table. In Cross System Product or VisualAge Generator, a collection of related data items structured as a two-dimensional array of columns and rows. Two kinds of tables can be defined:

- A Cross System Product or VisualAge Generator table refers to a member in the MSL that can be used for map validation, user messages, or as an internal data structure for an application. For VisualAge Generator 3.0, this table is defined as a VAGen part.
- A relational table refers to a table in a relational database accessed by SQL. A relational table is represented in Cross System Product or VisualAge Generator by a record with SQL row organization.

target environment. For Cross System Product or VisualAge Generator, the runtime environment for the generated application or program.

team programming. Development of a system, program, or application suite by a team of two or more programmers or application developers.

tool bar. In the **VisualAge Organizer**, **Composition Editor**, and **VAGen Parts Browser** windows, the strip of icons along the top of the freeform surface. The tool bar contains tools to help construct composite parts. These tools are also available through pull-down menus of the windows.

Transcript window. The main controlling window in Smalltalk.

U

unload. A system operation that removes a software component from an image by unlinking it. Contrast with *load*.

unexplodable. For the MSL Migration Assistance Tool, marking an application as *unexplodable* means that the parts within the application will not be automatically moved to a new ApplicationNode, even if the parts are associates of parts being migrated to a different application. Contrast with *explodable*.

V

VAGen export. In VisualAge Generator V3.0 or later, a command that copies one or more VAGen parts from a library to a serial file using external source format. Contrast with *VAGen import* and *File Out*. For Cross System Product or VisualAge Generator prior to V3.0, compare to *export*.

VAGen import. In VisualAge Generator V3.0 or later, a command that copies one or more VAGen parts from a serial file to a library. The serial file must be in external source format. Contrast with *VAGen export* and *File In*. For Cross System Product or VisualAge Generator prior to V3.0, compare to *import*.

VAGen part. In VisualAge Generator V3.0 or later, a portion of VisualAge Generator 4GL code that is associated with a Smalltalk method in an extension of its VAGen part class. It consists of a name and a complete definition. A VAGen part can be a data item, record, table, map, map group, program, process, statement group, PSB, or control information (linkage table, resource association file, generation options, bind command file, and linkage editor control statements). See also *VAGen part class*. Compare to *view*, *visual part*, and *method*. For Cross System Product or VisualAge Generator prior to V3.0, compare to *member*.

VAGen part class. In VisualAge Generator V3.0 or later, the type of code a VAGen part contains. The VAGen part classes are:

- VAGenDataItems
- VAGenRecords
- VAGenTables
- VAGenMaps
- VAGenMapGroups
- VAGenPrograms
- VAGenFunctions
- VAGenProcesses
- VAGenStatementGroups
- VAGenPSBs
- VAGenLinkages
- VAGenResources
- VAGenOptions
- VAGenBindControls
- VAGenLinkEdits

The VAGen part classes are extensions of a class. See also *VAGen part*. Compare to *class* and *extended class*. For Cross System Product or VisualAge Generator prior to V3.0, compare to *member type*.

validation. In Cross System Product or VisualAge Generator, the process that precedes the generation process. The validation process collects definition information for the object being generated. Validation also includes cross-checking to ensure the definitions are complete and correct.

variable. (1) A storage place within an object for a data element. The data element is an object, such as a number or date, stored as an attribute

of the containing object. (2) In VisualAge, a part that receives an identity at run time. A variable by itself contains no data or program logic; it must be connected such that it receives runtime identity from a part elsewhere in the application.

version. In the team programming environment an edition of a software component that cannot be changed. Each version has a version name, such as R4.0. Contrast with *edition* and *scratch edition*.

view. A composite visual part. A view can display and change the underlying nonvisual objects of an application. In VisualAge, views are both the end result of developing an application and the basic unit of composition of user interfaces. Compare to *visual part*.

visual part. A part that has a visual representation at run time. Visual parts, such as windows, push buttons, and entry fields, make up the user interface of an application. Compare to *view*. Contrast with *nonvisual part*.

W

window. A rectangular area of the screen with visible boundaries in which information is displayed. Windows can overlap on the screen, giving the appearance of one window being on top of another. See also *pane*.

work-in-progress. Members or parts that are currently undergoing change or testing of changes made to them.

workspace. A Java file that provides a development environment on an individual workstation. A workspace contains object instances, classes, and methods.

Special Characters

4GL. Parts written in the VisualAge Generator language including programs, records, tables, data items, map groups, maps, processes, statement groups, and PSBs. Views (GUIs) are not considered to be 4GL parts.

Index

Special Characters

/PROJECT generation option on
Java 11, 56

A

application
 assigning a manager 197, 337
 definition 208
 versioning 192, 332
ApplicationNode 250, 251, 310
associations
 4GL parts 44, 212
 GUIs 45, 213
 views 45, 213

B

business logic 98, 136, 278, 315
BW*Wizard 98, 278

C

code page conversion 118, 119, 297,
298
common parts 70, 71, 130, 250, 251,
310
configuration map
 adding a required map 194, 334
 assigning a manager 195, 335
 creating 193, 333
 definition 208
 naming conventions 225
 testing 196, 335
 versioning 195, 335
control information
 defining 35, 159, 199, 339
 general 42, 210
 naming conventions 10, 54, 173,
223
 storing in ENVY 10, 35, 43, 54,
159, 173, 199, 211, 223, 339
Cross System Product
 additional considerations 57,
236, 395
 collecting your source 117, 296
 control information 42, 210
 equivalent of advance
 command 86, 266
 migrating VSAM files 163, 343
 migration questionnaire 365

D

Data File Conversion utility 163,
344
data types, mismatched 234
dual maintenance 12, 57, 173, 235
duplicate names
 controlled duplicates 133, 312
 for business logic 136, 315
 unintended duplicates 133, 313
duplicate parts 49, 61, 69, 217, 241,
249

E

ENVY
 APIs 345
 characteristics 39, 207
 comparison to MSL 39, 207
 completing the setup 33, 151,
191, 331
 concepts 40, 208
 functional organization 44, 212
 part associations 44, 212
 part types 41, 209
 storing control information 10,
42, 54, 173, 210, 223
 storing parts 42, 210
explodable 131, 310

F

feature
 DDE support 295
 loading 295
 multimedia 295
functional organization 40, 44, 52,
208, 212, 219

G

generation
 /PROJECT on Java 11, 56
 changes to generation
 options 11, 56, 173, 225
 programs 160, 340
 project list part on Java 11, 57
 target environment
 considerations 79, 261
 VAGen runtime code 160, 340
group member 155, 196, 336
GUI
 associates moved to
 sandbox 259

GUI (*continued*)

 automatic conversion 229
 connection changes 232
 dual maintenance on 2.x and
 4.0 228
 inheritsCommSession 232
 log file 227
 migrating OS/2 category
 parts 227
 migration considerations 5, 49,
170, 226
 migration order 259
 mismatched data types 234
 moving to sandbox 259
 national characters 226
 Object Factory 234
 obsolete parts 232
 part associations 45, 213
 renamed features 229, 231
 renamed parts 229
 tasks after migrating 234
 View Wrapper 232

H

hptcnv40 119, 298
hptguicv.log 227
hprrules.nls 118, 297

I

inheritsCommSession 232

M

member type
 associations among 44, 212
 correspondence to VAGen part
 class 41, 209
migration
 collecting your source 117, 296
 completing 35, 159, 199, 339
 considerations
 assigning ownership 10, 53,
173, 220
 determining functional
 organization 52, 219
 establishing naming
 conventions 8, 50, 172, 217
 general 3, 47, 79, 167, 215,
259
 GUIs 5, 49, 170, 226
 resolving duplicate member
 names 49, 217

- migration (*continued*)
 - storing control information 10, 42, 54, 173, 210, 223
 - using configuration maps 225
 - using subapplications 6, 173, 221
 - VSAM files for Cross System Product 163, 343
- options for V3 to V4 Migration Tool 18, 178
- preparing for migration 15, 59, 175, 239
- status log for V3 to V4 Migration Tool 26, 185
- technique
 - all data items first 77, 257
 - programs and GUIs first 78, 259
 - records and tables first 78, 258
- mismatched data types 234
- missing parts 61, 71, 124, 135, 138, 139, 147, 241, 251, 303, 314, 317, 318, 327

MSL

- building directories using MSL Migration Assistance Tool 63, 121, 243, 300
- checking for unused parts 79, 260
- comparison to ENVY 39, 207
- concatenation 43, 211
- functional organization 44, 212
- member associations 44, 212
- member types 41, 209
- migrating production 76, 256
- migrating production and work-in-progress 75, 255
- migrating work-in-progress 35, 80, 161, 199, 261, 341
- production 75, 76, 255, 256
- scenario
 - complete set of MSLs for test 106, 285
 - delta MSLs for test 102, 282
 - general 85, 265
 - MSLs containing code from BW*Wizard 98, 278
 - MSLs containing code from VisualAge Generator Templates 98, 278
 - MSLs from marketing or other demonstrations 111, 289

MSL (*continued*)

- scenario (*continued*)
 - MSLs that contain unintended duplicates 97, 277
 - multiple subsystems with controlled duplicates 89, 269
 - multiple subsystems with no duplicates 87, 267
 - separate productions MSLs for each developer 92, 272
 - understanding diagrams 85, 265
 - understanding terminology 85, 265
- selecting MSLs for MSL Migration Assistance Tool 64, 125, 244, 304
- storing control information 42, 210
- storing members 42, 210
- work-in-progress 75, 80, 255, 261

- MSL Library Selection window 64, 125, 244, 304

MSL Migration Assistance Tool

- before starting migration 59, 239
- building MSL directories 63, 121, 243, 300
- changing required applications 322
- changing required packages 142
- checking consistency 141, 321
- checking relationships among applications 319
- checking relationships among packages 140
- collapsing a package 132
- collapsing an application 311
- committing to ENVY 72, 145, 252, 325
- controlling creation of ApplicationNodes 251, 310
- controlling creation of package.nodes 71, 130
- creating new application 308
- creating new package 129
- deleting a package.node 143
- deleting all applications 324
- deleting all packages 145
- deleting an ApplicationNode 323
- deleting one application 324
- deleting one package 144

MSL Migration Assistance Tool

(*continued*)

- determining the parts that are referenced 141, 320
- determining which programs are referenced 140, 319
- diagram 61, 241
- finding a part 137, 316
- handling duplicates 132, 312
- handling missing parts 139, 318
- identifying common code 70, 250
- identifying missing parts 71, 251
- listing missing parts 138, 317
- moving a part between applications 308
- moving a part between packages 129
- MSL Library Selection window 64, 244
- MSL Migration Part List window 62, 67, 242, 246
- normalizing required applications 322
- normalizing required packages 143
- other sandbox functions 71, 251
- overview 62, 242
- Part List Selection Criteria View window 65, 245
- performance 59, 239, 240
- preparing for migration 59, 239
- ready to migrate 59, 239
- refreshing for a fix test 346
- renaming a package 131
- renaming an application 311
- resetting from ENVY 72, 123, 252, 302
- resetting the sandbox 72, 123, 252, 302
- running 61, 115, 241, 293
- selecting and migrating VAGen parts 126, 305
- selecting MSLs 64, 125, 244, 304
- selecting parts for part list 65, 127, 245, 306
- selecting parts for sandbox 67, 127, 246, 306
- starting 62, 121, 242, 300
- updating required applications 321
- updating required packages 142
- using for work-in-progress MSLs 82, 264

MSL Migration Assistance Tool
(*continued*)
 VG Part Prerequisites View
 window 70, 250
 working in the sandbox 70, 250
MSL Migration Part List window
 Duplicate column 69, 249
 Last Migration Library
 Timestamp column 69, 73, 249,
 253
 other functions 70, 250
 selecting parts 67, 127, 246, 306
 special columns 68, 248
 Status column 68, 248

N

naming conventions 8, 50, 51, 53,
54, 79, 172, 217, 220, 223, 225, 259
not found parts 61, 69, 71, 124, 135,
138, 139, 147, 241, 248, 251, 303,
314, 317, 318, 327

O

Object Factory 234
ownership 10, 40, 53, 155, 173, 196,
208, 220, 336

P

package
 changing the owner 24, 156
 creating 31, 149
 definition 40
 naming conventions 8, 21, 50
 versioning 152
package.node 70, 71, 130
Part List Selection Criteria View
 window 65, 127, 245, 306
performance 59, 239
program
 definition 40, 208
 generating 160, 340
project
 changing the owner 155
 creating 31, 149
 definition 40
 versioning 153
project list part (PLP)
 creating 154
 definition 11, 57, 154
 generation 11, 57, 154
 organizing your code 7

Q

questionnaire
 Cross System Product 365
 VisualAge Generator 381

R

renamed features 229, 231
renamed parts 229

S

sandbox
 changing required
 applications 322
 changing required packages 142
 checking consistency 141, 321
 checking relationships among
 applications 319
 checking relationships among
 packages 140
 collapsing a package 132
 collapsing an application 311
 committing to ENVY 72, 145,
 252, 325
 controlling creation of
 ApplicationNodes 251, 310
 controlling creation of
 package.nodes 71, 130
 creating new application 308
 creating new package 129
 deleting a package.node 143
 deleting all applications 324
 deleting all packages 145
 deleting an
 ApplicationNode 323
 deleting one application 324
 deleting one package 144
 determining the parts that are
 referenced 141, 320
 determining which programs are
 referenced 140, 319
 finding a part 137, 316
 handling duplicates 132, 312
 handling missing parts 139, 318
 identifying common code 70,
 250
 identifying missing parts 71,
 251
 listing missing parts 138, 317
 moving a part between
 applications 308
 moving a part between
 packages 129
 moving parts to 67, 126, 246,
 305
 normalizing required
 applications 322
 normalizing required
 packages 143
 other functions 71, 251
 renaming a package 131

sandbox (*continued*)
 renaming an application 311
 resetting from ENVY 72, 123,
 252, 302
 updating required
 applications 321
 updating required packages 142
 working in 70, 250
subapplications 6, 173, 221
System Transcript window 345

T

TeamConnection, collecting your
 source code 117, 296

U

unexplodable 71, 131, 251, 310
user
 adding as group member 155,
 196, 336
 creating new user 116, 294
 setting current user 116, 294
utility, Data File Conversion 163,
344

V

V3 to V4 Migration Tool
 before starting migration 15, 175
 marking not migrated 29, 188
 migrating applications and
 configuration maps 24, 183
 migration options 18, 178
 migration status log 26, 185
 preparing for migration 15, 175
 ready to migrate 15, 175
 resetting 28, 187
VAGen
 changing from OS/2 to Windows
 NT 119, 298
VAGen Import 35, 75, 81, 161, 199,
255, 263, 341
VAGen part class
 changing ownership 156, 197,
 336
 correspondence to member
 type 41, 209
 definition 39, 207
 names of 41, 209
 releasing 151, 191, 331
 versioning 151, 191, 331
VAGen Parts Browser window 345
VG Part Prerequisites View
 window 70, 250
view
 definition 208
 packaging 340

- view (*continued*)
 - part associations 45, 213
 - releasing 151, 191, 331
 - versioning 151, 191, 331
- View Wrapper 232
- VisualAge Generator
 - additional considerations 12, 58, 117, 174, 237, 297
 - advance command 86, 266
 - collecting your source code 117, 297
 - control information 42, 210
 - hints and tips 345
 - loading a feature 295
 - migration questionnaire 381
 - setting preferences 294
 - starting 4.0 115, 293
 - using existing MSLs 117, 297
- VisualAge Generator Templates 7, 98, 171, 278
- VisualAge Organizer window 242, 345
- VSAM file 163, 343



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SH23-0267-00

