

IBM VisualAge TeamConnection



# Administrator's Guide

*Version 2.0*



IBM VisualAge TeamConnection



# Administrator's Guide

*Version 2.0*

## Second Edition (December 1997)

### Note

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

This edition applies to Version 2.0 of the licensed program IBM TeamConnection and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Order publications by phone or fax. The IBM Software Manufacturing Company 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.

A form for comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation  
Attn: Information Development  
Department T99B/Building 062  
P.O. Box 12195  
Research Triangle Park, NC, USA 27709-2195

You can fax comments to (919) 254-0206.

If you have comments about the product, address them to:

IBM Corporation  
Attn: Department TH0/Building 062  
P.O. Box 12195  
Research Triangle Park, NC, USA 27709-2195

You can fax comments to (919) 254-4914.

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 1992, 1995, 1996, 1997. All rights reserved.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Figures</b> . . . . .	xiii
<b>Notices</b> . . . . .	xv
<b>Trademarks</b> . . . . .	xvii
<b>About this book.</b> . . . .	xix
How this book is organized . . . . .	xix
Conventions . . . . .	xx
Tell us what you think . . . . .	xxi
<hr/>	
<b>Part 1. Introducing IBM VisualAge TeamConnection</b> . . . . .	<b>1</b>
<b>Chapter 1. An introduction to TeamConnection</b> . . . . .	<b>3</b>
TeamConnection definitions . . . . .	4
TeamConnection's client/server architecture . . . . .	4
TeamConnection database . . . . .	5
Interfaces . . . . .	5
Families . . . . .	6
Users and host lists. . . . .	6
Parts . . . . .	6
Components . . . . .	7
Releases. . . . .	7
Work areas . . . . .	8
Drivers . . . . .	9
Defects and features . . . . .	9
Processes . . . . .	9
Build . . . . .	11
Packaging . . . . .	11
<b>Chapter 2. Administrator Tasks</b> . . . . .	<b>13</b>
System Administrators . . . . .	13
Family Administrators . . . . .	14
Build Administrators. . . . .	14
<hr/>	
<b>Part 2. Installing the TeamConnection server</b> . . . . .	<b>15</b>
<b>Chapter 3. Installing TeamConnection for AIX</b> . . . . .	<b>19</b>
Hardware requirements . . . . .	20
Software requirements. . . . .	21
Preparing to install TeamConnection for AIX . . . . .	21
Updating TCP/IP files . . . . .	21
Preparing the syslog file . . . . .	23
Adding and mounting a CD-ROM file system . . . . .	24
Installing TeamConnection . . . . .	24
Installing TeamConnection components . . . . .	25
Creating a user account for your TeamConnection family . . . . .	26
Setting the environment variables for ObjectStore. . . . .	26
Verifying that the ObjectStore server is active . . . . .	26
Verifying the installation of TeamConnection . . . . .	27
What do you do now? . . . . .	30

<b>Chapter 4. Installing TeamConnection for HP-UX . . . . .</b>	<b>31</b>
Hardware requirements . . . . .	32
Software requirements. . . . .	33
Preparing to install TeamConnection for HP-UX . . . . .	33
Updating TCP/IP files . . . . .	33
Preparing the syslog file . . . . .	35
Adding and mounting a CD-ROM file system . . . . .	36
Installing TeamConnection . . . . .	36
Installing TeamConnection components . . . . .	37
Creating a user account for your TeamConnection family . . . . .	38
Setting the environment variables. . . . .	38
Verifying that the ObjectStore server is active . . . . .	38
Verifying the installation of TeamConnection . . . . .	39
What do you do now? . . . . .	42
 <b>Chapter 5. Installing TeamConnection for Solaris. . . . .</b>	 <b>43</b>
Hardware requirements . . . . .	43
Software requirements. . . . .	44
Preparing to install TeamConnection for Solaris . . . . .	44
Updating TCP/IP files . . . . .	45
Preparing the syslog file . . . . .	46
Installing TeamConnection . . . . .	47
 <b>Chapter 6. Installing TeamConnection for OS/2 . . . . .</b>	 <b>55</b>
Hardware requirements . . . . .	55
Software requirements. . . . .	57
Preparing to install TeamConnection for OS/2 . . . . .	58
Installing TeamConnection using SmartGuides . . . . .	59
Installing and configuring TeamConnection manually. . . . .	61
Installing TeamConnection . . . . .	61
Setting the environment variables. . . . .	62
Configuring and starting the database server . . . . .	63
Verifying the installation of TeamConnection . . . . .	64
What do you do now? . . . . .	67
 <b>Chapter 7. Installing TeamConnection for Windows . . . . .</b>	 <b>69</b>
Hardware requirements . . . . .	69
Software requirements. . . . .	70
Preparing to install TeamConnection for Windows. . . . .	71
Installing TeamConnection . . . . .	72
Installing the TeamConnection components . . . . .	73
Verifying TeamConnection installation . . . . .	74
Manual configuration . . . . .	74
Setting the environment variables. . . . .	75
Configuring and starting the ObjectStore server . . . . .	75
Creating a test family . . . . .	76
What do you do now? . . . . .	80
 <b>Chapter 8. Preparing a LAN installation for your client users . . . . .</b>	 <b>81</b>
Setting up a LAN installation for the client . . . . .	81
Updating the response file . . . . .	82
Adding IP address to the TCP/IP domain name server . . . . .	83
 <b>Chapter 9. Starting and stopping the servers . . . . .</b>	 <b>85</b>
Specifying the number of daemons to start . . . . .	85
Setting up the mail facility . . . . .	85

Starting the servers . . . . .	86
Starting the family server . . . . .	86
Starting servers from the TeamConnection group folder . . . . .	88
Starting servers from the Family Administrator GUI . . . . .	88
Starting build servers using teamcbld . . . . .	89
Caching and the build directories . . . . .	90
An MVS build server . . . . .	91
Starting a build agent for an MVS build server . . . . .	92
Creating build startup files . . . . .	94
Startup file for build servers . . . . .	94
Startup file for build agents . . . . .	95
Stopping the servers . . . . .	95
The family and notification servers . . . . .	95
A build server . . . . .	96
An MVS build server . . . . .	96

---

## Part 3. Designing and creating your TeamConnection environment . . . . . 97

<b>Chapter 10. Creating your TeamConnection family . . . . .</b>	<b>99</b>
Planning your families . . . . .	99
Creating a family . . . . .	99
Using the family settings notebook . . . . .	100
Family information . . . . .	101
Initial superuser . . . . .	102
Configurable fields . . . . .	103
Component processes . . . . .	103
Release processes . . . . .	104
User exits . . . . .	104
Security . . . . .	104
Authority groups . . . . .	105
Interest groups . . . . .	106
Creating an icon for an existing family . . . . .	107
For further information . . . . .	107
 <b>Chapter 11. Setting up your family structure . . . . .</b>	 <b>109</b>
Planning your components . . . . .	109
Organizing the component hierarchy . . . . .	109
Determining component ownership . . . . .	111
Naming the components . . . . .	111
Determining access to components . . . . .	112
Planning your releases . . . . .	112
Relating releases with components . . . . .	112
Selecting serial or concurrent development . . . . .	113
Controlling database growth . . . . .	113
Specifying separate databases . . . . .	115
Naming your releases . . . . .	115
Planning your processes . . . . .	115
Component processes . . . . .	116
Release processes . . . . .	116
How processes might change during development . . . . .	118
Using the driver subprocess . . . . .	119
Creating components and releases . . . . .	119
Creating components . . . . .	119
Creating releases . . . . .	120
Creating a new release from an old release . . . . .	121

<b>Chapter 12. Preparing for your users</b>	123
Planning for user IDs	123
Creating user IDs	124
Adding and modifying passwords	125
Planning for host lists	126
Creating host list entries	126
Planning for user access to TeamConnection data	127
What are the TeamConnection authority levels?	128
What are authority groups?	129
Creating or modifying authority groups	129
Granting authority to users	131
Granting or restricting access	132
Removing an entry from an access list	133
Planning for user notification	134
What are interest groups?	134
Creating or modifying interest groups	135
Working with notification lists	136
Displaying interest groups	137
Adding an entry to a notification list	137
Removing an entry from a notification list	138
 <b>Chapter 13. Working with configurable fields</b>	 141
Defining configurable field types	142
Changing or creating configurable fields	144
Creating and modifying configurable fields	145
Displaying configurable field properties	147
Changing report formats	147
The stanza report	147
The table report	150
 <b>Chapter 14. Configuring family processes</b>	 153
Modifying or creating configurable processes	153
 <b>Chapter 15. Providing user exits</b>	 155
Writing user exit programs	155
Environment file	161
Setting up user exits	162
Configuring user exit parameters	164
Sample user exit programs	165

---

## Part 4. Maintaining the TeamConnection server . . . . .167

<b>Chapter 16. Maintaining your TeamConnection environment</b>	169
Changing the age of defects and features	169
The age utility	169
The resetAge utility	170
Resolving TeamConnection errors	170
Using the error log	170
Using the audit log	170
Using the trace facility	176
Maintaining the TeamConnection database	177
osbackup	178
oschangedbref (oschange)	180
oscp	181
osrestore (osrestor)	182
ossevol	184



ossize . . . . .	185
ossvrping (ossvrpin). . . . .	186
ossvrstat (ossvrsta) . . . . .	186
osverifydb (osverify). . . . .	186
Using a raw file system (rawfs) database . . . . .	187
Creating a rawfs database . . . . .	187
Changing an existing file database to a rawfs database . . . . .	188
<b>Chapter 17. Migrating to TeamConnection version 2 . . . . .</b>	<b>189</b>
Migrating from TeamConnection version 1 to version 2 . . . . .	190
General guidelines . . . . .	190
Preparing the source server for migration . . . . .	191
Setting up the target server . . . . .	193
Performing the migration . . . . .	196
Preparing to administer your new database . . . . .	210
Importing fine-grained data into your migrated database . . . . .	211
Migrating from CMVC to TeamConnection version 2 . . . . .	212
General guidelines . . . . .	213
Preparing the source server for migration . . . . .	213
Setting up the target server . . . . .	218
Performing the migration . . . . .	222
Preparing to administer your new database . . . . .	229
Using the migration tools . . . . .	233
Migration tool variables . . . . .	235
Sample migration files . . . . .	236
Migrate command . . . . .	237
Report command. . . . .	242
Set command . . . . .	242
Select command . . . . .	243
Update command . . . . .	244
Delete command . . . . .	244
Describe command . . . . .	244
Exec command . . . . .	245
! command . . . . .	245
Quit command. . . . .	246
# command. . . . .	246
<b>Chapter 18. Monitoring family use. . . . .</b>	<b>247</b>
Using the license monitor. . . . .	247
How the license monitor counts users . . . . .	248
Using the tclcmn command . . . . .	248
Reporting highest uses . . . . .	249
Displaying a full use report . . . . .	250
Using the server daemon monitor. . . . .	253
Using the monitor command . . . . .	253
Monitoring the activity of the server daemons . . . . .	254
Detecting time-consuming requests . . . . .	255
Monitoring server daemon problems. . . . .	255

---

## Part 5. Installing and working with build servers . . . . .257

<b>Chapter 19. Basic build concepts . . . . .</b>	<b>259</b>
The physical structure of the build function . . . . .	259
The build object model. . . . .	261
Parent-child relationships in a build tree . . . . .	262
Working with a build tree . . . . .	263

Putting the pieces together . . . . .	265
<b>Chapter 20. Installing the build function . . . . .</b>	<b>267</b>
Preparing to install . . . . .	269
Installing build agents and processors on OS/2 and Windows NT . . . . .	270
Step 1: Starting the installation. . . . .	270
Step 2: Selecting installation options . . . . .	270
Step 3: Selecting the components for installation . . . . .	270
Step 4: Completing the installation . . . . .	271
Setting the environment variables. . . . .	271
Environment variables for a build agent . . . . .	271
Environment variables for an OS/2 build processor . . . . .	272
Installing build agents and processors on AIX and on HP-UX . . . . .	272
Installing build agents or processors on Windows NT . . . . .	272
Creating a build processor on MVS . . . . .	272
Customizing your JCL to use the LE/370 runtime libraries. . . . .	274
Environment variables for an MVS build processor . . . . .	275
<b>Chapter 21. Working with build scripts and builders . . . . .</b>	<b>277</b>
Creating a builder . . . . .	277
Writing a build script . . . . .	281
Passing parameters to a build script. . . . .	281
Writing a simple build script . . . . .	282
Writing an executable file for a build script . . . . .	283
Testing a build script . . . . .	284
Modifying the contents of a build script. . . . .	284
Putting a builder to work . . . . .	285
Removing a builder from a part . . . . .	286
Working with VisualAge C++ and Templates . . . . .	286
<b>Chapter 22. Working with MVS build scripts and builders . . . . .</b>	<b>287</b>
Creating a builder for MVS builds. . . . .	287
Writing an MVS build script . . . . .	291
File name conversions for MVS . . . . .	291
Passing parameters to an MVS build script . . . . .	292
TeamConnection syntax for MVS build scripts . . . . .	293
Supported JCL syntax . . . . .	293
Example of a build script for a C compile . . . . .	295
Example of a build script for a COBOL compile . . . . .	297
Example of a build script for a link . . . . .	298
<b>Chapter 23. Working with parsers . . . . .</b>	<b>301</b>
Creating a parser . . . . .	301
Putting a parser to work . . . . .	303
Removing a parser from a part . . . . .	304
Writing a parser command file . . . . .	305
<b>Chapter 24. Building an application: an example . . . . .</b>	<b>307</b>
Starting the build processors and build agents . . . . .	308
Creating builders and parsers . . . . .	309
Creating the build tree for the application . . . . .	309
Starting the build on the client . . . . .	313
Determining the build scope. . . . .	315
Adding the job to the job queue . . . . .	317
Picking up the work orders . . . . .	317
Putting the build processors to work. . . . .	317

Putting the build scripts to work . . . . .	317
Finishing the job and reporting the results to the user . . . . .	318
Monitoring the progress of a build . . . . .	318
Running a build in spite of errors . . . . .	319
Building all parts, regardless of build times . . . . .	319
Finding out which parts will be built . . . . .	320
Canceling a build. . . . .	320
More sample build trees . . . . .	321
Defining multiple outputs from a single build event . . . . .	321
Synchronizing the build of unrelated parts . . . . .	322

---

## Part 6. Using TeamConnection to package products . . . . . 323

<b>Chapter 25. Using TeamConnection to package a product . . . . .</b>	<b>325</b>
Setting up your build tree for packaging . . . . .	326
Setting up a build tree for the gather tool . . . . .	326
Setting up a build tree for the NVBridge tool. . . . .	328
Setting up a build tree for other distribution tools . . . . .	329
<b>Chapter 26. Using the Gather tool. . . . .</b>	<b>331</b>
Using the teamcpak command for the Gather tool. . . . .	332
Command line flags. . . . .	332
Examples of the teamcpak gather command . . . . .	333
Writing a package file for the Gather tool . . . . .	334
Syntax rules for a Gather package file . . . . .	334
<b>Chapter 27. Using the NVBridge tool . . . . .</b>	<b>339</b>
Using the teamcpak command for NVBridge. . . . .	340
Command line flags. . . . .	341
Examples of the teamcpak nvbridge command. . . . .	342
Writing a package file for NVBridge . . . . .	342
Syntax rules for an NVBridge package file . . . . .	343
Keywords for an NVBridge package file . . . . .	343
Problem determination for NVBridge. . . . .	351
NVBridge utilities. . . . .	352
FHPSTAT . . . . .	353
FHPOBDEL. . . . .	353
FHPOBMON . . . . .	353
FHPOBDIF . . . . .	354
FHPISCAT . . . . .	355
FHPICAT. . . . .	355
FHPUCAT . . . . .	356
FHPMCAT . . . . .	357
FHPVERIF . . . . .	357
FHPRQPUR . . . . .	358
FHPRQMON . . . . .	358
FHPTRVER. . . . .	359
FHPTRPUR . . . . .	360
<b>Chapter 28. Using the Tivoli Software Distribution packaging tool . . . . .</b>	<b>361</b>
Using the teamcpak command with Tivoli Software Distribution . . . . .	361
Command line flags. . . . .	362
Example of the teamcpak softdist command. . . . .	362
Writing a package file for Tivoli Software Distribution. . . . .	363
Syntax rules for a Tivoli Software Distribution package file . . . . .	363
Keywords for a Tivoli Software Distribution package file . . . . .	363

Problem determination for the Tivoli Software Distribution tool . . . . .	366
Sample package file . . . . .	366
<b>Appendix A. Family administration commands.</b> . . . . .	369
Creating a family database . . . . .	369
Creating an initial superuser for a family . . . . .	370
Creating or modifying authority groups . . . . .	371
Editing the authorit.ld file . . . . .	371
Reloading the authority table . . . . .	371
Creating or modifying interest groups . . . . .	372
Editing the interest.ld file . . . . .	372
Reloading the interest table . . . . .	373
Configuring component or release processes . . . . .	373
Editing the comproc.ld and relproc.ld files. . . . .	374
Reloading the configurable process tables . . . . .	374
Defining configurable field types . . . . .	375
Reloading the config table . . . . .	377
Creating and modifying configurable fields . . . . .	378
Creating configurable fields . . . . .	379
Creating configurable fields by copying from another family . . . . .	380
Updating configurable fields. . . . .	381
Changing report formats . . . . .	381
Setting up user exits . . . . .	384
Editing the userExit file . . . . .	384
Creating customized parameter lists. . . . .	385
<b>Appendix B. Configurable field types</b> . . . . .	387
Configurable field types . . . . .	387
Finding information about configurable fields . . . . .	390
<b>Appendix C. User exit parameters.</b> . . . . .	393
Parameters passed to user exit programs. . . . .	393
User exit parameter definitions. . . . .	413
<b>Appendix D. Environment Variables</b> . . . . .	425
Setting environment variables . . . . .	432
<b>Appendix E. TeamConnection NLS and DBCS considerations</b> . . . . .	433
Overview of TeamConnection NLS and DBCS support . . . . .	433
Supported locales . . . . .	434
Platforms supported by TeamConnection . . . . .	436
Locales supported by the ObjectStore database . . . . .	437
Characteristics and limitations of NLS and DBCS support. . . . .	437
No conversion of code points when exchanging data . . . . .	437
Exceptions to the handling of characters in TeamConnection. . . . .	439
All clients in the same host must use the same language (Intel only). . . . .	441
Untranslated strings that are visible to the users. . . . .	441
DBCS Limitations . . . . .	441
Installation, administration and runtime issues . . . . .	442
Installation issues related to NLS and DBCS . . . . .	442
Family administration issues . . . . .	444
Client runtime issues . . . . .	445
Miscellaneous topics that are specific to operating systems . . . . .	446
Problem resolution . . . . .	446
Topics related to the message catalog used by TeamConnection . . . . .	448
Setting and verifying the current locale in UNIX . . . . .	449

How to enter SBCS extended characters using an English keyboard . . . .	449
How to specify NLS settings for AIX GUI on aixterm/dtterm windows . . . .	449
Creating a Teamcgui resource file for AIX . . . . .	450
Using iconv to convert files to other code pages . . . . .	451
How to install additional locales for AIX . . . . .	452
How to change the keyboard for an AIX host . . . . .	452
How to install a non-English PC keyboard to an X-station . . . . .	453
How to start/shutdown the Xstation environment . . . . .	454
How to start the desktop environment to show the DBCS titles . . . . .	455
<b>Appendix F. Authority and notification for TeamConnection actions . . . .</b>	<b>457</b>
<b>Appendix G. Sample REXX execs, build scripts, and parsers . . . . .</b>	<b>473</b>
Sample REXX execs . . . . .	473
Sample build scripts . . . . .	476
Sample parsers . . . . .	477
Sample package files . . . . .	477
<b>Appendix H. Worksheets . . . . .</b>	<b>479</b>
Authority groups worksheet . . . . .	479
Interest groups worksheet . . . . .	483
Configurable processes worksheets . . . . .	488
<b>Customer support . . . . .</b>	<b>489</b>
<b>Bibliography . . . . .</b>	<b>491</b>
IBM VisualAge TeamConnection library . . . . .	491
Tool Builder's Development Kit. . . . .	491
TeamConnection Technical reports . . . . .	492
ObjectStore. . . . .	492
IBM Exchange library . . . . .	493
Related publications . . . . .	493
<b>Glossary . . . . .</b>	<b>495</b>
<b>Index . . . . .</b>	<b>503</b>



# Figures

1.	A sample TeamConnection client/server network . . . . .	5
2.	Sample of a component hierarchy. . . . .	7
3.	Parts, releases, and components . . . . .	8
4.	Create Family window . . . . .	66
5.	Create Family window . . . . .	79
6.	Family Settings notebook . . . . .	101
7.	A hierarchy representing product organization . . . . .	110
8.	A hierarchy showing parallel components . . . . .	110
9.	Components with more than one parent . . . . .	110
10.	A hierarchy showing component ownership . . . . .	111
11.	The release-component relationship . . . . .	113
12.	Using the driver subprocess . . . . .	119
13.	Create Components Window . . . . .	120
14.	Create Releases Window . . . . .	121
15.	Create User window. . . . .	125
16.	Add Host window . . . . .	126
17.	Granting authority to other users . . . . .	128
18.	The Authority Groups Settings page . . . . .	130
19.	Show Authority Actions window. . . . .	132
20.	Restrict Access window . . . . .	133
21.	Remove Access window . . . . .	134
22.	The Interest Groups Settings page . . . . .	136
23.	Show Interest Actions window . . . . .	137
24.	Add Notification window . . . . .	138
25.	Remove Notification window. . . . .	138
26.	The Configurable Fields Settings page . . . . .	143
27.	The Configurable Fields for Defects Settings page . . . . .	145
28.	Sample stanza report displayed after adding configurable fields. . . . .	148
29.	The Stanza View Format Settings page . . . . .	149
30.	Sample table report displayed after adding configurable fields . . . . .	150
31.	The Table View Format Settings page . . . . .	151
32.	The Component Processes Settings page. . . . .	154
33.	The User Exit settings page . . . . .	163
34.	User exit parameters settings page . . . . .	165
35.	Sample of an audit log file . . . . .	171
36.	The physical structure of TeamConnection . . . . .	260
37.	Sample build object model for msgcat.exe . . . . .	263
38.	The build tree for the hello application . . . . .	264
39.	Two versions of a build tree . . . . .	265
40.	Build agents and processors on separate machines . . . . .	268
41.	Build agents on a single machine . . . . .	269
42.	Create Builder window . . . . .	278
43.	Matching environment values . . . . .	280
44.	Modify Part Properties window . . . . .	285
45.	Modify Part Properties window . . . . .	286
46.	Create Builder window . . . . .	288
47.	Matching environment values . . . . .	290
48.	A JCL fragment for an MVS compile . . . . .	296
49.	A JCL fragment converted to a build script . . . . .	297
50.	Create Parser window . . . . .	302
51.	Modify Part Properties window . . . . .	304
52.	Modify Part Properties window . . . . .	305
53.	Sample build tree. . . . .	307

54. Sample build object model for msgcat.exe . . . . .	308
55. Create Parts window . . . . .	310
56. Create Parts window . . . . .	311
57. Modify Part Properties window . . . . .	312
58. Connect Parts window . . . . .	312
59. The build tree display . . . . .	313
60. Build Parts window . . . . .	313
61. Build tree showing build times . . . . .	316
62. The build tree for robot.dll. . . . .	321
63. The build tree for robot.app . . . . .	322
64. Part of the build tree for robot.app . . . . .	326
65. Adding the gather step to the build tree. . . . .	328
66. Adding the NVBridge step to the build tree . . . . .	329
67. The chfield display with configurable field information filled in. . . . .	380
68. Sample report format after adding configurable fields. . . . .	383



---

## 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, USA 10594.

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 Site Counsel, IBM Corporation, P.O. Box 12195, 3039 Cornwallis Road, Research Triangle Park, NC 27709-2195, USA. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement.

This document is not intended for production use and is furnished as is without any warranty of any kind, and all warranties are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

IBM may change this publication, the product described herein, or both. These changes will be incorporated in new editions of the publication.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.



---

## Trademarks

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

AIX	MVS/ESA
BookManager	MVS/XA
Common User Access	NetView
Courier	Operating System/2
CUA	OS/2
C/370	TeamConnection
DB/2	Tivoli
IBM	VisualAge
MVS	XGA

The following terms are trademarks of other companies:

**ObjectStore**

ObjectStore Design, Inc.

**UNIX** X/Open Company Limited

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

Solaris and the Sun logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.



---

## About this book

This book is part of the documentation library supporting the IBM VisualAge TeamConnection licensed programs. It is written for persons who need to perform the following tasks:

- Plan for the installation of TeamConnection.
- Install the TeamConnection servers or prepare for LAN-based installation of TeamConnection clients.
- Set up TeamConnection databases, and administer TeamConnection families.
- Create and administer configurable fields and user exits.
- Maintain TeamConnection databases or migrate a database to TeamConnection version 2.
- Install and work with TeamConnection build servers.

The user who needs to install only a TeamConnection client should follow the installation instructions in the *Getting Started with the TeamConnection Clients* book.

This book is available in PDF format. Because production time for printed manuals is longer than production time for PDF files, the PDF files may contain more up-to-date information. The PDF files are located on the installation CD in directory path `softpubs\enu` (`softpubs/en_us` in UNIX). To view these files, you need a PDF reader such as Acrobat.

---

## How this book is organized

This book contains the following sections:

**“Part 1. Introducing IBM VisualAge TeamConnection” on page 1** presents an overview of the IBM VisualAge TeamConnection product. The information in this section should be read and understood by everyone who is going to work with TeamConnection.

**“Part 2. Installing the TeamConnection server” on page 15** explains how to install the TeamConnection server software. It contains one chapter for each supported platform. For instructions on installing the TeamConnection client software, refer to *Getting Started with the TeamConnection Clients*. This section also contains information on starting and stopping the servers.

**“Part 3. Designing and creating your TeamConnection environment” on page 97** is intended for the family administrator who needs to plan for how the IBM VisualAge TeamConnection product is going to be used in the company's development environment. After the planning stage, the family administrator will use this section to learn how to do TeamConnection administrative tasks. Before reading this section, you should be familiar with the TeamConnection terminology and concepts presented in “Part 1. Introducing IBM VisualAge TeamConnection” on page 1.

**“Part 4. Maintaining the TeamConnection server” on page 167** contains information on maintaining your TeamConnection database, monitoring family use, and migrating your database from CMVC or TeamConnection version 1 to TeamConnection version 2.

**Note to Version 3 customers**

Documentation for Version 3 migration utilities is not yet available.

“**Part 5. Installing and working with build servers**” on page 257 tells how to install and use the TeamConnection build function.

“**Part 6. Using TeamConnection to package products**” on page 323 tells how to package TeamConnection parts for distribution.

This book also contains several appendixes providing more information for performing TeamConnection administrative tasks and worksheets that can help you plan your TeamConnection families.

Information on customer service, a glossary, and a bibliography are included at the back of this book.

---

## Conventions

This book uses the following highlighting conventions:

- *Italics* are used to indicate the first occurrence of a word or phrase that is defined in the glossary. They are also used for information that you must replace.
- **Bold** is used to indicate items on the GUI.
- Monospace font is used to indicate exactly how you type the information.
- File names follow Intel conventions: **mydir\myfile.txt**. AIX and HP-UX users should render this file name **mydir/myfile.txt**.

Tips or platform specific information is marked in this book as follows:



Shortcut techniques and other tips



IBM VisualAge TeamConnection for OS/2



IBM VisualAge TeamConnection for Windows 3.1



IBM VisualAge TeamConnection for Windows/NT



IBM VisualAge TeamConnection for Windows 95



IBM VisualAge TeamConnection for AIX



IBM VisualAge TeamConnection for HP-UX



IBM VisualAge TeamConnection for Solaris

---

## Tell us what you think

In the back of this book is a comment form. Please take a few moments to tell us what you think about this book. The only way for us to know if you are satisfied with our books or if we can improve their quality is through feedback from customers like you.





---

## Part 1. Introducing IBM VisualAge TeamConnection

<b>Chapter 1. An introduction to TeamConnection</b>	3
TeamConnection definitions	4
TeamConnection's client/server architecture	4
TeamConnection database	5
Interfaces	5
Families	6
Users and host lists	6
Parts	6
Components	7
Releases	7
Work areas	8
Drivers	9
Defects and features	9
Processes	9
Build	11
Packaging	11
 <b>Chapter 2. Administrator Tasks</b>	 13
System Administrators	13
Family Administrators	14
Build Administrators	14



---

## Chapter 1. An introduction to TeamConnection

TeamConnection provides an environment and tools to make software development run smoothly, whether your development team is small or large. Using TeamConnection, you can communicate with and share data among team members to keep up with the many tasks in the development life cycle, from planning through maintenance.

What does TeamConnection do for you? It takes care of the following:

- *Configuration management*: the process of identifying, organizing, managing, and controlling software modules as they change over time. This includes controlling access to your software modules and providing notification to team members as software modules change.
- *Release management*: the logical organization of objects that are related to an application. The release provides a logical view of objects that must be built, tested, and distributed together. Releases are versioned, built, and packaged.
- *Version control*: the tracking of relationships among the versions of the various parts that make up an application. Version control enables you to build your product using stable levels of code, even if the code is constantly changing. It provides control over which changes are available to everyone and, optionally, allows more than one developer at a time to update a part.
- *Change control*: the controlling of changes to parts that are stored in TeamConnection. TeamConnection keeps track of any part changes you make and the reasons you make them. Your development team can build releases with accuracy and efficiency, even as the parts evolve. The product ensures that the change process is followed and that the changes are authorized. After changes are made, it allows you to integrate the changes and build the application. TeamConnection tracks all changes to the parts across multiple products and environments.  
  
The *change control process* is configurable. Your team can decide how strict the change control should be, from loose to very tight. You can also adjust the level of control as you move through a development cycle.
- *Build support*: the function that enables you to define the structure of your application and then to create it within TeamConnection from your input parts. Independent steps in a build can run in parallel on different servers, thus reducing your build time. You can build applications for platforms in addition to the one TeamConnection runs on—currently, you can use TeamConnection to build applications on AIX, HP-UX, OS/2, Windows NT, Windows 95, Solaris, and MVS.
- *Packaging support*: the preparation of your application for electronic distribution to other users.

TeamConnection provides an open information model for sharing data between a set of integrated tools using TeamConnection. This object-based information model enables an extensible architecture, thus ensuring continued support for new versions of existing tools, as well as new tools that are brought into the repository environment. This support includes the previously mentioned standard services across all objects stored in the information model. TeamConnection's information model and tool builder's functions are provided separately in the Tool Builder's Development Kit. See the bibliography at the back of this book for more information.

IBM Exchange for OS/2, which is a feature of TeamConnection, provides the functions necessary to migrate existing model information into TeamConnection. For more information about IBM Exchange, refer to the *IBM Exchange User's Guide*

This chapter defines the basic terms and concepts you need to make the most of TeamConnection. Read this chapter first; then decide which information you need next:

Topic and description	Page
Installing the TeamConnection server	17
• Installing TeamConnection for AIX	
• Installing TeamConnection for HP	
• Installing TeamConnection for Solaris	
• Installing TeamConnection for OS/2	
• Installing TeamConnection for Windows	
• Preparing a LAN installation for your client users	
• Recovering from installation errors	
• Installing the build function	
• Starting and stopping the servers	
Designing and creating your TeamConnection environment:	98
• Planning for and creating families	
• Preparing for users	
• Configuring fields	
• Configuring <i>processes</i>	
• Providing user exits	
Maintaining TeamConnection:	168
• Setting up the mail facility	
• Changing the age of <i>defects</i> and <i>features</i>	
• Resolving TeamConnection errors	
• Maintaining the database	

---

## TeamConnection definitions

The following definitions are in logical order rather than alphabetical. The *User's Guide* provides additional information about these terms.

## TeamConnection's client/server architecture

Figure 1 on page 5 is an example of a network of TeamConnection clients and servers.

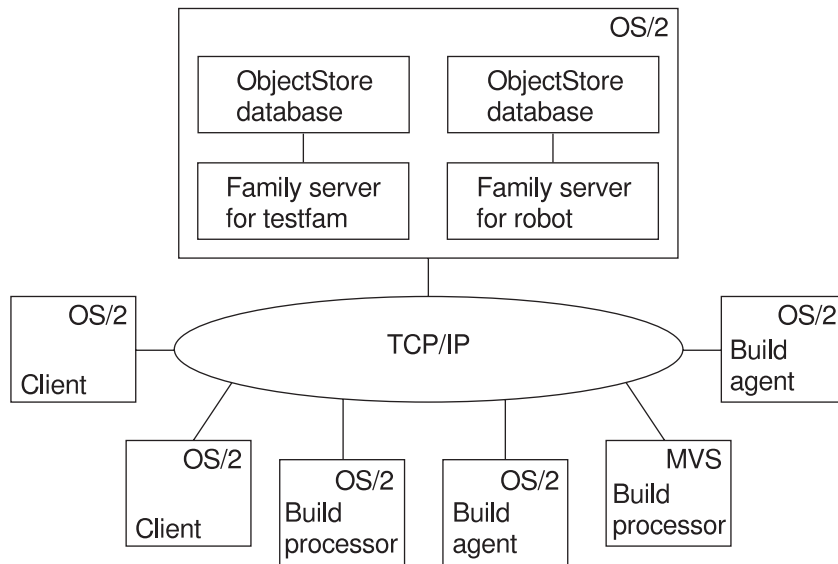


Figure 1. A sample TeamConnection client/server network

TeamConnection *family servers* control all data within the TeamConnection environment. Data stored in a family server's database includes:

- Text objects, such as source code and product documentation
- Binary objects, such as compiled code
- Modeled objects that are stored in the information model by tools such as VisualAge Generator
- Other TeamConnection objects that are metadata about the other objects

*Build agents* and *build processors* work together to perform product builds. We sometimes refer to the combination of a build agent and its connected build processor as a build server.

A TeamConnection *client* gives team members access to the development information and parts stored on the database server.

## TeamConnection database

TeamConnection is built on Object Design, Inc.'s ObjectStore database. The TeamConnection server communicates with the ObjectStore database through an ObjectStore server.

## Interfaces

TeamConnection provides the following interfaces that you can use to access data:

- A graphical user interface based on Common User Access (CUA).
- A command line interface that lets you type TeamConnection commands from a prompt or from within TeamConnection
- A web client, that you access through your web browser.

You can use any interface to do your TeamConnection work, or you can switch among them.

## Families

A *family* represents a complete and self-contained collection of TeamConnection users and development data. Data within a family is completely isolated from data in all other families. One family cannot share data with another.

See “Chapter 11. Setting up your family structure” on page 109

## Users and host lists

Users are given access to the TeamConnection development data in a specific family through their *user IDs*. Each family has at least one *superuser*, who has privileged access to the family. The superuser gives other users the *authority* to perform some set of *actions* on particular data. Depending on the authority granted to a user, that user might in turn be able to grant some equal or lesser level of authority to other users. However, the ability to grant authority for some actions is reserved to the superuser. There are no actions which the superuser cannot perform.

For host-based authentication, each user ID is associated with a *host list*, which is a list of client machine addresses from which the user can access TeamConnection when using that ID.

A single user can access TeamConnection from multiple systems or logins. Likewise, a single system login can act on behalf of multiple users. The set of authorized logins for a TeamConnection user ID makes up the user's host list.

It is also possible to authenticate users through the use of passwords, either in place of host lists, or as an alternative form of authentication. See “Chapter 12. Preparing for your users” on page 123

## Parts

TeamConnection *parts* are objects that users and tools store in TeamConnection. They include text objects, binary objects, and modeled objects. These parts can be stored by the user or the tool, or they can be generated from other parts, such as when a linker generates an executable file. Parts can also be groupings of other TeamConnection objects for building and distribution, or simply for convenient reference. Common part actions include the following:

### Create

To store a part from your workstation on the server; from that time on, TeamConnection keeps track of all changes made to the part. Or, to create a part to use as a place holder to store the output of a build.

### Check out

To get a copy of a part so that you can make changes to it.

### Check in

To put the changed part back into TeamConnection.

### Extract

To get a copy of the part *without* making changes to the *current version* in TeamConnection.

### Edit

To change a part from within TeamConnection using a specified editor.



application; that is, all parts that must be built, tested, and distributed together. Each time a release is changed, a new version of the release is created. Each version of the release points to the correct version of each part in the release.

Each part in TeamConnection is managed by at least one component and contained in at least one release. One release can contain parts from many components; a component can span several releases. Figure 3 shows the relationships between parts, the releases that contain them, and the components that manage them.

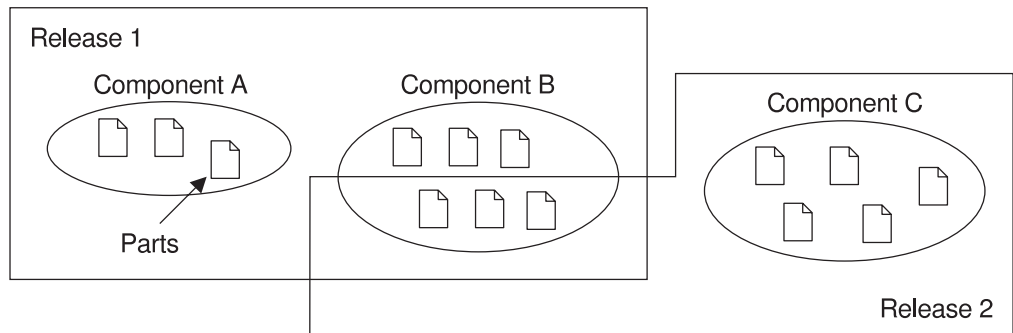


Figure 3. Parts, releases, and components

Each time a new development cycle begins, you can define a separate release. Each subsequent release of an application can share many of the same parts as its predecessor. Thus maintenance of an older release can progress at the same time as development of a newer one. Each release follows a process by which defects and features are handled.

See “Planning your releases” on page 112 for more information about releases.

## Work areas

A release contains the latest “official” version of each of its parts. As users check parts out of the releases, update them, and then check them back in, TeamConnection keeps track of all of these changes, even when more than one user updates the same part at the same time. To make this possible, TeamConnection uses something called a *work area*.

A work area is a logical temporary work space that enables you to isolate your work on the parts in a release from the official versions of the parts. You can check parts out to a work area, update them, and build them without affecting the official version of the parts in the release. After you are certain that your changes work, you *integrate* the work area with the release (or *commit* the driver that the work area is a member of, if you are using the driver subprocess). The integration makes the parts from your work area the new official parts in the release.

You can do the following with work areas:

- Check out parts from a release
- Update any or all of the checked-out parts
- Get the latest copies of the parts in the release, including any changes integrated by other users
- Get the latest copies of the parts in another work area



- *Freeze* the work area, making a snapshot of the parts as they exist at a particular instant in case you need to return to it later
- Build the parts in the work area
- Move all parts back into the release by integrating the work area

## Drivers

A driver is a collector for work areas. You create drivers associated with specific releases so that you can exercise greater control over which work areas are integrated into the release and commit the changes from multiple work areas simultaneously.

When a work area is added to a driver, it is called a *driver member*. A single work area can be a member of more than one driver. By making a work area part of a driver, you associate the parts changed in relation to that work area with the specified driver. These parts must be members of the release associated with the driver.

Drivers enable you to place the following controls over work area integrations:

- Define and monitor prerequisite and corequisite work areas to ensure that mutually dependent changes are integrated in proper order.
- Monitor and resolve conflicting changes to the same part (if you use concurrent development).
- Restrict access to driver members so that they can be changed only by users with proper authority.

## Defects and features

A defect is a record of a problem to be fixed. A feature is a record of a request for a functional addition or enhancement. Both are associated with a work area, and both follow the processes defined for the component and release that are associated with the work area. TeamConnection tracks both objects through their life cycles as developers change and commit parts.

You can use defects and features to record problems and design changes for things other than the products you are developing under TeamConnection control. For example, you can use defects to record information about personnel problems, hardware problems, or process problems. You can use features to record proposals for process improvements and hardware design changes.

## Processes

An application changes over time as developers add features or correct defects. TeamConnection controls these changes according to the *processes* you choose for your application's components and releases. A process enforces a specific level of control to part changes and ensures that actions occur in a specified order.

Two separate types of processes are defined: component processes, which can be different for each component within a family, and release processes, which apply to all activities associated with a given release. Component or release processes are built from a number of lower-level processes, or *subprocesses*, that are included with the TeamConnection product.

A defect or feature written against a component moves through successive *states* during its life cycle. The TeamConnection actions that you can perform against it depend on its current state. The component processes define these actions. You can require users to do some, all, or none of the following for tracking defects and features:

**dsrFeature**

Design, size, and review changes to be made for features

**verifyFeature**

Verify that the features have been implemented correctly

**dsrDefect**

Design, size, and review fixes to be made for defects

**verifyDefect**

Verify that the fixes work

At the release level you can require some, all, or none of the following subprocesses:

**track** This subprocess is TeamConnection's way of relating all part changes to a specific defect or feature and a specific release. Each work area gathers all the parts modified for the specified defect or feature in one release and records the status of the defect or feature. The work area moves through successive states during its life cycle. The TeamConnection actions that you can perform against a work area depend on its current state.

You must use the *track subprocess* if you want to use any of the other release subprocesses.

**approval**

This subprocess ensures that a designated *approver* agrees with the decision to incorporate changes into a particular release and electronically signs a record. As soon as approval is given, the changes can be made.

**fix** This subprocess ensures that as users check in parts associated with a work area, an action is taken to indicate that they have completed their portion. When everyone is done, the owner of the *fix record* (usually the component owner) can change the fix record to complete. The parts are then ready for integration.

**driver** A *driver* is a collection of all the work areas that are to be integrated with each other and with the unchanged parts in the release at a particular time. The driver subprocess allows you to include these changes incrementally so that their impact can be evaluated and verified before additional changes are incorporated. Each work area that is included in a driver is called a *driver member*.

**test** The test subprocess guarantees that testing occurs prior to verifying that the fix is correct within the release.

TeamConnection is shipped with several predefined processes. If these do not apply to your organization, you can configure your own processes by defining different combinations of subprocesses.

See "Planning your processes" on page 115 for an explanation of TeamConnection states.

## Build

The TeamConnection build function automates the process of building individual parts or entire applications, both in the work group LAN environment and on an enterprise server. This function enables you to reliably and repeatedly build the same output from the same inputs. You can also build different outputs from the same inputs for different environments.

You start a build against an output part that has an associated *builder*. A builder is an object that describes how to translate input parts to get the desired output, such as a linker or compiler. An input part might have an associated parser, which determines the dependencies for the input parts in a build.

The build function does the following:

- Tracks build times of inputs and outputs so that it builds only those parts that are out of date themselves or that have out of date dependants. You can also force a build regardless of the build times.
- Enables you to spread the build over multiple machines running at the same time or into multiple processes running on a single machine, such as on MVS.

## Packaging

*Packaging* is any of the steps necessary to distribute software and data onto the machines where they are to be used. TeamConnection includes two tools that you can use to automate the electronic distribution of TeamConnection-managed software and data:

### Gather

An automated data mover for server or file transfer-based distribution

### NVBridge

A bridge utility that automates the installation and distribution of software or data using IBM NetView Distribution Manager/2 as the distribution vehicle

### NVBridge

A tool that supports automated distribution between a single NetView DM/2 CC server and its LAN-connected CC clients. It also supports remote distribution to APPC-connected NetView DM/2 servers and mainstream servers.

For more information, see “Part 6. Using TeamConnection to package products” on page 323



---

## Chapter 2. Administrator Tasks

This chapter briefly describes the tasks that a TeamConnection administrator performs. Administrator's responsibilities vary widely according to the needs of your development environment and the size and complexity of your network. The tasks explained in this book can be performed by a single system administrator or by two or more administrators. One way to distinguish administrators' roles is by function, as follows:

### **System administrator**

Has *superuser* access to the family server and database administration access to the database management system. This administrator is responsible for the following:

- Installing and maintaining the server
- Maintaining and backing up the database used by TeamConnection

### **Family administrator**

Has superuser access to the family server and database administration access to the database management system. This administrator is responsible for the following:

- Planning and configuring TeamConnection for one or more families
- Managing user access to one or more families
- Maintaining one or more families

### **Build administrator**

This administrator is responsible for the following:

- Setting up and maintaining build servers
- Planning for builds
- Creating builders and parsers
- Starting and stopping build agents and processors
- Defining *pools*
- Monitoring build performance
- Creating driver members
- Committing and completing drivers
- Extracting releases
- Packaging and distributing applications

---

## System Administrators

System administrators are responsible for installing the TeamConnection server software, creating user accounts for TeamConnection families, updating network and services configuration files for TCP/IP and socket addresses used by families and build agents, preparing the TeamConnection client software for LAN installation (if your installation plans to make the client software available over a LAN), starting and stopping the servers, and maintaining the TeamConnection databases. Instructions for completing these tasks are in "Part 2. Installing the TeamConnection server" on page 15.

One particularly important function of a system administrator is maintaining the TeamConnection databases. Your TeamConnection database needs to be backed up regularly using the ObjectStore utilities `osbackup` and `osrestore`. These and other database utilities are explained in "Chapter 16. Maintaining your

TeamConnection environment” on page 169. Please be sure you read this chapter and carefully follow the maintenance procedures included in it.

---

## Family Administrators

If your TeamConnection environment includes more than one family, you might consider assigning one family administrator to each family. Family administrators are responsible for planning and creating the component structure and releases to be used in your family, configuring the processes to be used for the components and releases, creating user IDs and managing their access to the family, and creating configurable fields and user exits.

“Part 3. Designing and creating your TeamConnection environment” on page 97 contains instructions for completing each of these tasks.

---

## Build Administrators

TeamConnection provides build environments for most of its platforms. If you have a large and complex project, or your development efforts require you to build on multiple platforms, it may be beneficial for you to assign a build administrator for TeamConnection or for each TeamConnection family. Build administrators are responsible for installing and maintaining the build servers, configuring your build environment, creating build scripts and parsers, monitoring build performance, and customizing packaging and distribution scripts.

“Part 5. Installing and working with build servers” on page 257 explains how to create and maintain a build environment.

---

## Part 2. Installing the TeamConnection server

<b>Chapter 3. Installing TeamConnection for AIX</b>	19
Hardware requirements	20
Software requirements	21
Preparing to install TeamConnection for AIX	21
Updating TCP/IP files	21
Preparing the syslog file	23
Adding and mounting a CD-ROM file system	24
Installing TeamConnection	24
Installing TeamConnection components	25
Creating a user account for your TeamConnection family	26
Setting the environment variables for ObjectStore	26
Verifying that the ObjectStore server is active	26
Verifying the installation of TeamConnection	27
Step 1 — Configuring TeamConnection environment variables	27
Step 2 — Ensure that the TeamConnection client is accessible	28
Step 3 — Creating testfam	28
Step 4 — Start the family server	29
Step 5 — Verify TeamConnection installation	29
Step 6 — Using the TeamConnection client	30
What do you do now?	30
<b>Chapter 4. Installing TeamConnection for HP-UX</b>	31
Hardware requirements	32
Software requirements	33
Preparing to install TeamConnection for HP-UX	33
Updating TCP/IP files	33
Preparing the syslog file	35
Adding and mounting a CD-ROM file system	36
Installing TeamConnection	36
Installing TeamConnection components	37
Creating a user account for your TeamConnection family	38
Setting the environment variables	38
Verifying that the ObjectStore server is active	38
Verifying the installation of TeamConnection	39
Step 1 — Configuring TeamConnection environment variables	39
Step 2 — Ensure that the TeamConnection client is accessible	40
Step 3 — Creating testfam	40
Step 4 — Start the family server	41
Step 5 — Verifying TeamConnection installation	41
Step 6 — Using the TeamConnection client	42
What do you do now?	42
<b>Chapter 5. Installing TeamConnection for Solaris</b>	43
Hardware requirements	43
Software requirements	44
Preparing to install TeamConnection for Solaris	44
Updating TCP/IP files	45
Preparing the syslog file	46
Installing TeamConnection	47
Installing TeamConnection components	48
Creating a user account for your TeamConnection family	48
Setting the environment variables for ObjectStore	49
Verifying that the ObjectStore server is active	49

Verifying the installation of TeamConnection . . . . .	49
What do you do now? . . . . .	52
<b>Chapter 6. Installing TeamConnection for OS/2 . . . . .</b>	<b>55</b>
Hardware requirements . . . . .	55
Software requirements. . . . .	57
Preparing to install TeamConnection for OS/2 . . . . .	58
Installing TeamConnection using SmartGuides . . . . .	59
Installing and configuring TeamConnection manually. . . . .	61
Installing TeamConnection . . . . .	61
Step 1: Starting the installation. . . . .	61
Step 2: Selecting installation options . . . . .	61
Step 3: Selecting the components for installation . . . . .	61
Step 4: Completing the installation . . . . .	62
Setting the environment variables. . . . .	62
Configuring and starting the database server . . . . .	63
Verifying the installation of TeamConnection . . . . .	64
Step 1. Create your test family. . . . .	64
Step 2. Start the family server . . . . .	66
Step 3. Verify TeamConnection installation . . . . .	67
Step 4. Starting the TeamConnection client . . . . .	67
What do you do now? . . . . .	67
<b>Chapter 7. Installing TeamConnection for Windows . . . . .</b>	<b>69</b>
Hardware requirements . . . . .	69
Software requirements. . . . .	70
Preparing to install TeamConnection for Windows. . . . .	71
Installing TeamConnection . . . . .	72
Installing the TeamConnection components . . . . .	73
Verifying TeamConnection installation . . . . .	74
Manual configuration . . . . .	74
Setting the environment variables. . . . .	75
Configuring and starting the ObjectStore server . . . . .	75
Creating a test family . . . . .	76
Verifying that the database server is active . . . . .	76
Creating testfam . . . . .	77
Starting the family server . . . . .	79
Starting the TeamConnection client . . . . .	80
What do you do now? . . . . .	80
<b>Chapter 8. Preparing a LAN installation for your client users . . . . .</b>	<b>81</b>
Setting up a LAN installation for the client . . . . .	81
Updating the response file . . . . .	82
Adding IP address to the TCP/IP domain name server . . . . .	83
<b>Chapter 9. Starting and stopping the servers . . . . .</b>	<b>85</b>
Specifying the number of daemons to start . . . . .	85
Setting up the mail facility . . . . .	85
Starting the servers . . . . .	86
Starting the family server. . . . .	86
Using teamcd . . . . .	86
Starting servers from the TeamConnection group folder . . . . .	88
Starting servers from the Family Administrator GUI . . . . .	88
Starting build servers using teamcbd . . . . .	89
Caching and the build directories . . . . .	90
An MVS build server . . . . .	91



The build cache data sets . . . . .	91
Customizing the cache data set space attribute . . . . .	92
Starting a build agent for an MVS build server . . . . .	92
Creating build startup files . . . . .	94
Startup file for build servers . . . . .	94
Startup file for build agents . . . . .	95
Stopping the servers . . . . .	95
The family and notification servers . . . . .	95
A build server . . . . .	96
An MVS build server . . . . .	96

This section explains how to install the TeamConnection server software. It contains one chapter for each supported platform. For instructions on installing the TeamConnection client software, refer to *Getting Started with the TeamConnection Clients*

This section also contains information on starting and stopping the servers.



---

## Chapter 3. Installing TeamConnection for AIX

This chapter lists the hardware and software that are required before you can install and use the TeamConnection for AIX product. It also describes the tasks that you must do before installation, and provides detailed instructions for installing and configuring the TeamConnection family server and client.

### **Note:**

If you have previously installed CMVC and want to migrate from CMVC to TeamConnection version 2, you need to install TeamConnection version 2 on a separate machine and migrate your CMVC database to TeamConnection. See “Chapter 17. Migrating to TeamConnection version 2” on page 189 for instructions on migrating from CMVC to TeamConnection.

After you install the TeamConnection code in its directory structure (represented by `$TC_HOME`), please keep in mind the following guidelines:

- Do not remove directories, files, or symbolic links from `$TC_HOME`, unless you have uninstalled the product.
- Do not change the file permissions or the ownership of the directories or files in `$TC_HOME`.
- Use `$TC_HOME` only to store the TeamConnection code. Do not use it as the home directory for users and do not use it as the home directory for TeamConnection families.
- If this is your first installation of TeamConnection, accept the default values suggested in the installation instructions and scripts.

---

## Hardware requirements

The following are hardware requirements for the TeamConnection AIX **family server** and **build server** machines:

*Table 1. Hardware Requirements for an AIX TeamConnection Server*

<b>Family server</b>	<ul style="list-style-type: none"><li>• Processor: IBM RS/6000 with PowerPC architecture (recommended), such as g30. PowerServer architecture can be used.</li><li>• Pointing device: A mouse or other pointing device</li><li>• Monitor: Any X11 graphics display supported by the processor</li><li>• Memory: 128 MB memory minimum; more may be needed according to number of users and database size</li><li>• Disk space:<ul style="list-style-type: none"><li>– 500 MB for operating system and prerequisites</li><li>– 200 MB for user data (1+ GB recommended)</li><li>– 200 MB for TeamConnection server code</li><li>– 65 MB for TeamConnection documentation</li><li>– 140 MB for ObjectStore</li><li>– 100 MB for temporary space</li><li>– Amount recommended by operating system for swapper space</li></ul></li><li>• Communications support: Any token-ring or Ethernet local area network (LAN) adapter card that supports TCP/IP and is supported by the workstation</li><li>• A CD-ROM drive (internal or external) for installation of the product</li></ul>
<b>Build server</b>	<ul style="list-style-type: none"><li>• Processor: IBM RS/6000 with PowerPC architecture (recommended), such as g30. PowerServer architecture can be used.</li><li>• Pointing device: A mouse or other pointing device</li><li>• Monitor: Any X11 graphics display supported by the processor</li><li>• Memory: 64 MB memory minimum (128 MB recommended); more may be needed according to the compilers and linkers used</li><li>• Disk space:<ul style="list-style-type: none"><li>– 500 MB for operating system and prerequisites</li><li>– 200 MB for TeamConnection server code</li><li>– 140 MB for ObjectStore</li><li>– 100 MB for temporary space</li><li>– Amount recommended by operating system for swapper space</li></ul></li><li>• Communications support: Any token-ring or Ethernet local area network (LAN) adapter card that supports TCP/IP and is supported by the workstation</li><li>• A CD-ROM drive (internal or external) for installation of the product</li></ul>

The following are hardware requirements for the TeamConnection AIX **client** machine:

Table 2. Hardware Requirements for an AIX TeamConnection Client

<b>Processor</b>	IBM RS/6000 with PowerPC architecture (recommended), such as 43P. PowerServer architecture can be used.
<b>Pointing device</b>	A mouse or other pointing device.
<b>Monitor</b>	Any X11 graphics display supported by the processor.
<b>Memory</b>	32 MB memory minimum.
<b>Disk space</b>	300 MB for operating system and prerequisites, 60 MB for TeamConnection client code, Amount recommended by operating system for swapper space.
<b>Communications support</b>	Any token-ring or Ethernet local area network (LAN) adapter card that supports TCP/IP and is supported by the workstation.
<b>CD-ROM drive</b>	A CD-ROM drive (internal or external) for installation of the product.

---

## Software requirements

The software requirements for the AIX **family server**, **client**, and **build server** are the following:

Table 3. Software Requirements for TeamConnection for AIX

<b>Server</b>	AIX version 4.2.1 (part number 5765-393) or a later release of version 4 that includes TCP/IP, and the xIC runtime
<b>Client</b>	<ul style="list-style-type: none"><li>• AIX version 4.2.1 (part number 5765-393) or a later release of version 4 that includes TCP/IP</li><li>• A Web browser such as Netscape Navigator to display the help panels for the GUI</li></ul>
<b>Build server</b>	AIX version 4.2.1 (part number 5765-393) or a later release of version 4 that includes TCP/IP, and the xIC runtime

---

## Preparing to install TeamConnection for AIX

This section explains what you need to do before installing TeamConnection for AIX. To install TeamConnection, you should be an AIX system administrator who can log in as user root.

Use the System Management Interface Tool (smit) to create a user account, update the hosts and services files, and add and mount a file system. This tool presents a menu-driven environment for system administration tasks. For more information on smit, refer to your AIX operating system documentation.

## Updating TCP/IP files

Before you install the TeamConnection code, you must have the correct version of TCP/IP installed on your workstation. See “Software requirements” for the communications software requirements for your operating system.

After TCP/IP is installed, update your TCP/IP services and hosts files.

Do the following steps. You can update these files using smit. In many installations, a name server is used instead of /etc/hosts. Also, many installations distribute /etc/services. smit can be used to make the necessary updates.

1. Update the services file, which is located in `/etc/services`.

Include the family name and port address of the TeamConnection server. The port address can be any 4-digit number, as long as it does not already exist in your services file. You might want to ask your TCP/IP administrator to assign you a number.

Type the following entry in your services file:

```
# TeamConnection servers
testfam    ffff/tcp    # port address for the TeamConnection test family
```

**Notes:**

- a. For this installation, replace `ffff` with an appropriate port address.
  - b. Follow the line with a carriage return.
2. Update the TCP/IP hosts file, which is located in `/etc/hosts`.  
Add the following:
    - IP address.
    - Server name.
    - Alias name of the TeamConnection family server, which is your family name.  
For this initial installation of TeamConnection, the family name is `testfam`.

The following is an example of the entry you would type in your hosts file:

```
9.12.345.67    teamserv.company.com    testfam
```

**Note:** Follow the line with a carriage return. You can use the `hostname` command to get the name of the server.

3. Do the following to verify that the hosts file is specified correctly:
  - Type `host family_name`, where `family_name` is the name of your TeamConnection family. The information returned should match the number and name specified in your hosts file entry. For example, using the entry given in the previous step, the system response would be as follows:  

```
teamserv.company.com = 9.12.345.67
```
  - Type `host ip_address`, where `ip_address` is the IP address of your machine. The information returned should match the number and name specified in your hosts file entry. For example, again using the entry given in the previous step, the system response would be as follows:  

```
9.12.345.67 = teamserv.company.com
```

If you do not receive the expected response, contact your TCP/IP administrator to solve the problem.

4. Do the following to verify that you can connect to your TeamConnection family:
  - a. At a prompt, type `ping testfam`.
  - b. Press `Ctrl+C` to end the command.

If you receive information that is similar to the following, you can successfully connect to your TeamConnection family:

```
PING teamserv.company.com: 56 data bytes
64 bytes from 1.23.457.78: icmp_seg:0. time=0. ms
64 bytes from 1.23.456.78: icmp_seg:1. time=0. ms
64 bytes from 1.23.456.78: icmp_seg:2. time=0. ms
```

If you receive the message `unknown host testfam`, you cannot connect to the family. Verify that the data you entered in the hosts and services files is correct, and then try the command again. If you still do not get the correct response, contact your TCP/IP administrator to solve the problem.

## Preparing the syslog file

TeamConnection and database errors are recorded in the syslog file. The information recorded in this file will help you resolve problems when they occur.

Before TeamConnection can record these errors, you must activate the syslog daemon. Do the following to activate the syslog daemon:

1. The syslog file is located in `/var/spool/syslog`. If the log file does not exist, type the following touch command to create it. When you create the log file, follow the directions for setting permissions for your operating system.

```
touch /var/spool/syslog
chmod 666 /var/spool/syslog
chown root /var/spool/syslog
chgrp system /var/spool/syslog
```

If you don't want to use `chmod 666`, you can use `chmod a=rw /var/spool/syslog` or `chmod ugo=rw /var/spool/syslog`.

The command `chown root:system` is a short way to do the following:

```
chown root /var/spool/syslog
chgrp system /var/spool/syslog
```

2. Add the following lines to the `/etc/syslog.conf` file:

```
*.warning /var/spool/syslog
```

3. Type the following to stop and then restart the syslog daemon:

```
stopscr -s syslogd
startsrc -s syslogd
```

4. Type the following command to verify that the syslog daemon is running:

```
ps -ef | grep syslog
```

If the daemon is running, you will see one process for syslogd.

5. Do the following to verify that the syslog daemon is able to write to the syslog file:

- a. Log in as another user ID.
- b. Type `su root` and an incorrect password to add an error message to the syslog.

To quickly look at the last ten lines of the syslog, type the following command, substituting the proper name of your syslog file for *syslogFileName*:

```
tail -n 10 syslogFileName
```

If the syslog file is configured properly, it will contain a message similar to the following:

```
Apr 19 10:21:45 hostname su: BAD SU from userid to root at /dev/pts/3
```

6. To clean up the syslog, type the following command, substituting the proper name of your syslog file for *syslogFileName*:

```
cp /dev/null syslogFileName
```

7. Monitor the syslog file at regular intervals so that you can perform any maintenance or problem resolution.

## Adding and mounting a CD-ROM file system



**Quick start for experts:** Mount the CD-ROM as **/cdrom**. Use the **/cdrom/tcinst.ksh** command to install the TeamConnection components you want.

To install the TeamConnection code from a CD-ROM, you must first add and mount a CD-ROM file system. Follow the instructions in this section to complete this task.

Do the following to add and mount a CD-ROM file system:

1. Insert the installation CD into the CD-ROM drive.
2. Log in as user **root** or type **su - root**.
3. To create a directory for the CD-ROM, type **mkdir /cdrom**.
4. Enter **smit** to add a CD-ROM file system.
5. Select **System Storage Management (Physical & Logical Storage)**.
6. Select **File Systems**.
7. Select **Add / Change / Show / Delete File Systems**.
8. Select **CDROM File Systems**.
9. Select **Add a CDROM File System**.
10. Select a device name, such as **cd0**. The CD-ROM file system device name must be unique. You may need to delete a previously defined CD-ROM file system if there is a conflict.
11. Type **/cdrom** in the **Mount Point** field.
12. Select **OK**, or press Enter if you are using the smit ASCII interface.
13. Return to the previous smit level, **System Storage Management (Physical & Logical Storage)**.
14. Select **File Systems**.
15. Select **Mount a File System**.
16. Select **/dev/cd0** for **File System name**.
17. Select **/cdrom** for **Directory over which to mount**.
18. Select **cdvfs** for **Type of file system**.
19. Select **yes** for **Mount as a READ-ONLY system**.
20. Select **OK**, or press Enter if you are using the smit ASCII interface.
21. Exit smit.

Now you are ready to install the appropriate TeamConnection components.

---

## Installing TeamConnection

This section provides step-by-step instructions for the initial installation of TeamConnection, and gives instructions for verifying the installation.

This section is intended for the administrator who does the initial installation of TeamConnection for your organization. To perform the installation tasks, you need to be an AIX system administrator who can log in as user **root** and create user



accounts. This installation process assumes that you are installing all of the following components on the same workstation:

- TeamConnection family server (including the database)
- Client
- Build server

If you do not follow these instructions as written, the verification step on page “Step 5 — Verify TeamConnection installation” on page 29 might fail.

The time required for installing this program varies depending on your workstation. In general, installation of the product takes about 45 minutes.

After completing the installation and verification instructions in this section, you will have a family called testfam. Consider this your test family, and use it to verify that TeamConnection is working properly. This family is configured with the default information that IBM ships.

You can use testfam to explore and learn about TeamConnection. Using testfam with the shipped default information will help you determine how to customize TeamConnection to fit your development needs.

**Note:** When installing only a client, follow the instructions in *Getting Started with the TeamConnection Clients*

The following is a list of the tasks you do to install TeamConnection:

Task	Page
Install the TeamConnection components	26
Create a user account for the TeamConnection family	26
Set the environment variables	26
Verify that the ObjectStore server is active	26
Verify the installation	“Verifying the installation of TeamConnection” on page 27
Start the TeamConnection client	“Step 6 — Using the TeamConnection client” on page 30

## Installing TeamConnection components



**Quick start for experts:** Mount the CD-ROM as **/cdrom**. Use the **/cdrom/tcinst.ksh** command to install the TeamConnection components you want.

**Note:** If you have previously installed CMVC and want to migrate from CMVC to TeamConnection version 2, you need to install TeamConnection version 2 on a separate machine and migrate your CMVC database to TeamConnection. See “Chapter 17. Migrating to TeamConnection version 2” on page 189 for instructions on migrating from CMVC to TeamConnection.

After you have added and mounted a CD-ROM, do the following to install the TeamConnection components:

1. Log in with root authority.
2. Change the directory to **/cdrom**.
3. Run the installation script `tcinst.ksh` by typing **`./tcinst.ksh`**.  
In AIX, the files on a CD-ROM are always shown in lowercase, and they need to be specified in lowercase.
4. Follow the instructions in the installation script to install the TeamConnection server and client.
5. After the installation is complete, type the following commands:  

```
cd ..  
umount /cdrom
```
6. Eject the CD.

## Creating a user account for your TeamConnection family

Each TeamConnection family requires a separate user account.

Log in as root user ID and create a new user account for your family. This can be performed using the AIX SMIT tool.

## Setting the environment variables for ObjectStore

Now that TeamConnection for AIX is installed, you must set the environment variables for the root user ID. These variables are needed to manage the database. Do the following steps:

1. Add the following variables to the `.profile` for the root user to start and shut down the ObjectStore database:  

```
OS_ROOTDIR=/usr/lpp/ODI/OS4.0/cset  
export OS_ROOTDIR  
PATH=$PATH:$OS_ROOTDIR/bin:$OS_ROOTDIR/lib  
export PATH  
LIBPATH=$OS_ROOTDIR/lib  
export LIBPATH
```
2. Exit the root user ID and log in again as root to update the environment variables.

## Verifying that the ObjectStore server is active

**Note:** If ObjectStore is installed, then the ObjectStore server will be started by the installation script.

Before you create your test family, do the following to verify that the ObjectStore server is active:

1. From a prompt, type:  

```
ps -ef | grep os
```

2. Verify that there is a process called `osserver`.

If you have already created a family, then you might see the process `oscmgr4` (for the ObjectStore cache manager).

You can start and stop these ObjectStore processes as follows:

- To start the ObjectStore server, type the following command:

```
osserver
```

- To shut down the ObjectStore cache manager, type the following command:

```
oscmshtd
```

- To shut down the ObjectStore server, type the following command. Substitute the host name of your server for *<hostname>*.

```
ossvrshd -f <hostname>
```

## Verifying the installation of TeamConnection

Before you can verify that TeamConnection is installed correctly, you must create your family. When you create your family, you are loading default information shipped by IBM and creating a superuser ID. A superuser ID is required so that at least one person has privileged access to the family to perform special tasks, such as creating other user IDs.

This section leads you through the following procedures to create a test family called *testfam*.

1. Configuring the TeamConnection environment variables in the `.profile`.
2. Verifying that the TeamConnection client is accessible.
3. Creating `testfam`.
4. Starting the family server.
5. Verifying TeamConnection installation.
6. Using the TeamConnection client.

### Step 1 — Configuring TeamConnection environment variables

Environment variables in the `.profile` of the family account need to be configured before you create a TeamConnection family.

It is recommended that the `.profile` for the family be based on the sample `profile.family`. This profile uses all the necessary environment variables to create and use a TeamConnection family.

1. Log in as the user account for the TeamConnection family.
2. Use the following commands to copy the sample `profile.family` as your new `.profile`, substituting the location where TeamConnection is installed for *\$TC\_HOME* and the identifier for the national language version of TeamConnection you are using (such as `en_us`) for *\$LANG*.

```
mv $HOME/.profile $HOME/.profile.original
cp $TC_HOME/install/$LANG/profile.family $HOME/.profile
```

3. Customize all appropriate entries in the new `.profile`.

Edit the new `.profile`, read the entire file, modify the values as appropriate, and uncomment variables necessary for your operating system.

If you are using RAWFS databases, create a placeholder database in `TC_DBPATH` using the name `.tcd`. For example, if your `TC_DBPATH` is `/teamc/testfam` and your family name is `testfam`, create a file called `/teamc/testfam/testfam.tcd`. It does not matter what is in the file, as long as it

exists. The locator file (/teamc/testfam/testfam.loc for the previous example) determines the actual database host and location. Also, make sure the environment variable OS\_DIRMAN\_HOST is not set.

**Note:** If you are using the CDE environment, edit .dtpfile to enable the account to run .profile after .dtpfile. CDE stands for Common Desktop Environment and is the default for AIX 4 as well as many other versions of UNIX. CDE is the standard of the Open Group (c). For instructions, see the header of the sample profile.

4. You will likely want to restrict access to this account. Be sure that your .profile (and .dtpfile in the CDE environment) have no visibility outside of the user account. To restrict access to the account, issue the following command:

```
chmod u=rw,go= .profile .dtpfile
```

5. Log off and log in as the TeamConnection family to activate the changes to the new profile.

## Step 2 — Ensure that the TeamConnection client is accessible

Because the verification step uses the TeamConnection client, you need to ensure that it is accessible:

1. Type the following TeamConnection command, which does not require connection to a working TeamConnection family.

```
teamc report -testClient
```

2. If the teamc command is not found, then ensure that the TC\_HOME and PATH variables are properly set in the .profile.
3. If the teamc command is found, then verify that the rop portion of the output is something like this, assuming that the LANG variable is en\_US:

```
Product Version:                2.0.4
Message catalog language:       English
Environment variables:
Variable:                       Setting:
-----
TC_BECOME                       <your TeamConnection user id>
TC_FAMILY                       <your TeamConnection family name>
```

4. If you see that the entry for "Message catalog language" has the value English (Internal), then the LANG or NLSPATH variables are not properly set and the TeamConnection client is using its internal messages. It is strongly recommended that the external message catalog be used.

## Step 3 — Creating testfam

To create a test family to be used for installation verification, follow these steps.

1. Ensure that you have logged in with the ID of the family account.
2. Enter the following command from a command prompt:

```
dbcreate
```

This command creates the testfam database and an initial TeamConnection user with superuser authority. This user is the family administrator.

After testfam is created, you should be able to see the following subdirectories and files in the family account:

### authorit.Id

The default authority groups for the family

**cfgField**

A subdirectory containing default configurable field information

**comproc.ld**

The default component process for the family

**config**

A subdirectory containing default user exit information for the family and security information

**config.ld**

The default configurable field types for the family

**interest.ld**

The default interest groups for the family

**queue**

A subdirectory for the messages to be sent by the notify daemon

**relproc.ld**

The default release processes for the family

**schema.adb**

Metadata (definitions of objects such as users, parts, and releases) for the family

**testfam.tcd**

The family database

**Step 4 — Start the family server**

To start the testfam family server, type the following command from a command prompt. Use of this command assumes that the path statements and environment variables have been set properly.

```
teamcd testfam
```

**Step 5 — Verify TeamConnection installation**

The following installation verification procedure runs a command file called univunix, which does the following:

- Ensures that all TeamConnection daemons are stopped (using stopteamc)
- Starts the family daemon and a build agent and processor (using teamcd and temporary TCP/IP socket numbers)
- Runs a suite of TeamConnection actions

Running this command file requires that you have a full build environment set up, including VisualAge C++. "Chapter 20. Installing the build function" on page 267 explains how to install the build function. If you have not installed the build function or VisualAge C++, please omit this procedure.

To verify that TeamConnection installed correctly, type the following at a prompt in the directory where TeamConnection is installed, and then press Enter.

```
univunix
```

For more information on this command, type the following at a command prompt:

```
univunix -help
```

The univunix utility creates a file called univunix.log. Review univunix.log for a completion code of zero on the commands it runs. If all commands run successfully, then TeamConnection was installed and configured correctly.

When you are finished using the test data, you can restore the original database by issuing the following commands:

```
mv <family>.tcd <family>.tcd.test #or you can delete the test family
mv <family>.db0 <family>.tcd
```

The original database, <family>.db0, is valid only as long as you do not perform any family configuration against the current <family>.tcd. Once you perform any family configuration, please delete <family>.db0.

## Step 6 — Using the TeamConnection client

Before you start the client GUI for the first time, do the following:

1. Copy the sample initial tasks list for the main window. You will need to do this only once:

```
cp $TC_HOME/nls/cfg/$LANG/teamc20.ini $HOME
chmod u+w $HOME/teamc20.ini
```

2. Start the TeamConnection GUI by issuing the following command:

```
teamcgui &
```

You can issue TeamConnection line commands from the family account. To see a list of all users, for example, issue the following command:

```
teamc report -view users -where 1=1
```

## What do you do now?

You now have your initial family installed and running. As mentioned earlier in this chapter, you or someone else in your organization can use this family to verify that TeamConnection is working properly. You can also use this family to explore and learn about TeamConnection, and test configuration changes, user exits, automated backup utilities, and automated build and packaging events.

The remaining chapters of this book provide instructions for installing build servers and instructions for starting and stopping the servers.

To learn more about planning for and creating your TeamConnection production family, see “Part 3. Designing and creating your TeamConnection environment” on page 97 .

---

## Chapter 4. Installing TeamConnection for HP-UX

This chapter lists the hardware and software that are required before you can install and use the TeamConnection for HP-UX server. It also describes the tasks that you must do before installation. For hardware and software requirements and for instructions on installing the client software, refer to *Getting Started with the TeamConnection Clients*.

### Note:

If you have previously installed CMVC and want to migrate from CMVC to TeamConnection version 2, you need to install TeamConnection version 2 on a separate machine and migrate your CMVC database to TeamConnection. See “Chapter 17. Migrating to TeamConnection version 2” on page 189 for instructions on migrating from CMVC to TeamConnection.

After you install the TeamConnection code in its directory structure (represented by \$TC\_HOME), please keep in mind the following guidelines:

- Do not remove directories, files, or symbolic links from \$TC\_HOME, unless you have uninstalled the product.
- Do not change the file permissions or the ownership of the directories or files in \$TC\_HOME.
- Use \$TC\_HOME only to store the TeamConnection code. Do not use it as the home directory for users and do not use it as the home directory for TeamConnection families.
- If this is your first installation of TeamConnection, accept the default values suggested in the installation instructions and scripts.

---

## Hardware requirements

The following are hardware requirements for the TeamConnection HP-UX **family server** and **build server** machines:

*Table 4. Hardware Requirements for an HP TeamConnection Server*

<b>Family server</b>	<ul style="list-style-type: none"><li>• Processor: Any HP 9000 Series 700 or 800 workstation</li><li>• Pointing device: A mouse or other pointing device</li><li>• Monitor: Any X11 graphics display supported by the processor</li><li>• Memory: 128 MB memory minimum; more may be needed according to number of users and database size</li><li>• Disk space:<ul style="list-style-type: none"><li>– 500 MB for operating system and prerequisites</li><li>– 200 MB for user data (1+ GB recommended)</li><li>– 200 MB for TeamConnection server code</li><li>– 65 MB for TeamConnection documentation</li><li>– 140 MB for ObjectStore</li><li>– 100 MB for temporary space</li><li>– Amount recommended by operating system for swapper space</li></ul></li><li>• Communications support: Any token-ring or Ethernet local area network (LAN) adapter card that supports TCP/IP and is supported by the workstation</li><li>• A CD-ROM drive (internal or external) for installation of the product</li></ul>
<b>Build server</b>	<ul style="list-style-type: none"><li>• Processor: Any HP 9000 Series 700 or 800 workstation</li><li>• Pointing device: A mouse or other pointing device</li><li>• Monitor: Any X11 graphics display supported by the processor</li><li>• Memory: 64 MB memory minimum (128 MB recommended); more may be needed according to the compilers and linkers used</li><li>• Disk space:<ul style="list-style-type: none"><li>– 500 MB for operating system and prerequisites</li><li>– 200 MB for TeamConnection build server code</li><li>– 140 MB for ObjectStore</li><li>– 100 MB for temporary space</li><li>– Amount recommended by operating system for swapper space</li></ul></li><li>• Communications support: Any token-ring or Ethernet local area network (LAN) adapter card that supports TCP/IP and is supported by the workstation</li><li>• A CD-ROM drive (internal or external) for installation of the product</li></ul>

The following are hardware requirements for the TeamConnection HP-UX **client** machine:

*Table 5. Hardware Requirements for an HP TeamConnection Client*

<b>Processor</b>	Any HP 9000 Series 700 or 800 workstation.
<b>Pointing device</b>	A mouse or other pointing device.
<b>Monitor</b>	Any X11 graphics display supported by the processor.



Table 5. Hardware Requirements for an HP TeamConnection Client (continued)

<b>Memory</b>	32 MB memory minimum.
<b>Disk space</b>	500 MB for operating system and prerequisites, 60 MB for TeamConnection client code, Amount recommended by operating system for swapper space.
<b>Communications support</b>	Any token-ring or Ethernet local area network (LAN) adapter card that supports TCP/IP and is supported by the workstation.
<b>CD-ROM drive</b>	A CD-ROM drive (internal or external) for installation of the product.

---

## Software requirements

The software requirements for the HP-UX **family server**, **client**, and **build server** are the following:

Table 6. Software Requirements for TeamConnection for HP

<b>Server</b>	<ul style="list-style-type: none"><li>• HP-UX version 10.01, which includes TCP/IP</li><li>• C++ runtime version B.004 or later</li></ul>
<b>Client</b>	HP-UX version 10.01, which includes TCP/IP
<b>Build server</b>	<ul style="list-style-type: none"><li>• HP-UX version 10.01, which includes TCP/IP</li><li>• C++ runtime version B.004 or later</li></ul>

---

## Preparing to install TeamConnection for HP-UX

This section explains what you need to do before installing TeamConnection for HP-UX. To install TeamConnection, you should be an HP-UX system administrator who can log in as user root.

This procedure uses the sam command, which starts the System Administration Management tool. This tool presents an environment for system administration tasks. For more information on sam, refer to your HP-UX operating system documentation.

## Updating TCP/IP files

Before you install the TeamConnection code, you must have the correct version of TCP/IP installed on your workstation. See “Software requirements” for the communications software requirements for your operating system.

After TCP/IP is installed, update your TCP/IP services and hosts files.

Do the following steps. You can update these files using sam. In many installations, a name server is used instead of /etc/hosts. Also, many installations distribute /etc/services. You can use the sam tool to make the necessary updates.

1. Update the services file, which is located in /etc/services.

Include the family name and port address of the TeamConnection server. The port address can be any 4-digit number, as long as it does not already exist in your services file. You might want to ask your TCP/IP administrator to assign you a number.

Type the following entries in your services file:

```
# TeamConnection servers
testfam    ffff/tcp    # port address for the TeamConnection test family
```

**Notes:**

- a. For this installation, replace *ffff* with an appropriate port address.
  - b. Follow the line with a carriage return.
2. Update the TCP/IP hosts file, which is located in */etc/hosts*.

Add the following:

- IP address.
- Server name.
- Alias name of the TeamConnection family server, which is your family name. For this initial installation of TeamConnection, the family name is *testfam*.

The following is an example of the entry you would type in your hosts file:

```
9.12.345.67    teamserv.company.com    testfam
```

**Note:** Follow the line with a carriage return. You can use the *hostname* command to get the name of the server.

3. Do the following to verify that the hosts file is specified correctly:

**Note:** The *tchost.hp* utility is available from the *misc* directory on the TeamConnection CD-ROM.

- Type *tchost.hp family\_name*, where *family\_name* is the name of your TeamConnection family. The information returned should match the name specified in your hosts file entry. For example, using the entry given in the previous step, the system response would be as follows:  

```
teamserv.company.com
```
- Type *tchost.hp ip\_address*, where *ip\_address* is the IP address of your machine. The information returned should match the name specified in your hosts file entry. For example, again using the entry given in the previous step, the system response would be as follows:  

```
teamserv.company.com
```

If you do not receive the expected response, contact your TCP/IP administrator to solve the problem.

**Note:** If the servers are defined in the domain name server, then you can use the UNIX utility *"nslookup"* instead.

4. Do the following to verify that you can connect to your TeamConnection family:
  - a. At a prompt, type *ping testfam*.

If you receive information that is similar to the following, you can successfully connect to your TeamConnection family:

```
PING teamserv.company.com: 56 data bytes
64 bytes from 1.23.457.78: icmp_seg:0. time=0. ms
64 bytes from 1.23.456.78: icmp_seg:1. time=0. ms
64 bytes from 1.23.456.78: icmp_seg:2. time=0. ms
```

5. Press *Ctrl+C* to end the command.

If you receive the message *unknown host testfam*, you cannot connect to the family. Verify that the data you entered in the hosts and services files is correct, and then try the command again. If you still do not get the correct response, contact your TCP/IP administrator to solve the problem.

## Preparing the syslog file

TeamConnection and database errors are recorded in the syslog file. The information recorded in this file will help you resolve problems when they occur.

Before TeamConnection can record these errors, you must activate the syslog daemon. Do the following to activate the syslog daemon:

1. The syslog file is located in `/var/adm/syslog`. If the log file does not exist, type the following touch command to create it. When you create the log file, follow the directions for setting permissions for your operating system.

```
touch /var/adm/syslog
chmod 666 /var/adm/syslog
chown root /var/adm/syslog
chgrp system /var/adm/syslog
```

If you don't want to use `chmod 666`, you can use `chmod a=rw /var/adm/syslog` or `chmod ugo=rw /var/adm/syslog`.

The command `chown root:system` is a short way to do the following:

```
chown root /var/spool/syslog
chgrp system /var/spool/syslog
```

2. Add the following line to the `/etc/syslog.conf` file:

```
*.warning /var/adm/syslog
```

3. Type the following to stop and then restart the syslog daemon:

```
kill -HUP cat /etc/syslog.pid
```

4. Type the following command to verify that the syslog daemon is running:

```
ps -ef | grep syslog
```

If the daemon is running, you will see one process for `syslogd`.

5. Do the following to verify that the syslog daemon is able to write to the syslog file:

- a. Log in as another user ID.
- b. Type `su root` and an incorrect password to add an error message to the syslog.

To quickly look at the last ten lines of the syslog, type the following command, substituting the proper name of your syslog file for *syslogFileName*:

```
tail syslogFileName
```

If the syslog file is configured properly, it will contain a message similar to the following:

```
Apr 19 10:21:45 hostname su: BAD SU from userid to root at /dev/pts/3
```

6. To clean up the syslog, type the following command, substituting the proper name of your syslog file for *syslogFileName*:

```
cp /dev/null syslogFileName
```

7. Monitor the syslog file at regular intervals so that you can perform any maintenance or problem resolution.

## Adding and mounting a CD-ROM file system



**Quick start for experts:** Mount the CD-ROM as **/CDROM**. Use the **"/CDROM/TCCDUTIL.KSH;1"** command to install the TeamConnection components you want.

**Note:** Enter these names and commands exactly as shown, using uppercase letters and quotes, when shown.

To install the TeamConnection code from a CD-ROM, you must first add and mount a CD-ROM file system. Follow the instructions in this section to complete this task.

Do the following to add and mount a CD-ROM file system:

1. Insert the installation CD into the CD-ROM drive.
2. Log in as user **ROOT** or type **su - root**.
3. Issue the **mount** command to determine if the CD-ROM file system is mounted and operational. Verify that **/CDROM** (or equivalent) is listed; if, not, then do the following:
  - a. To create a directory for the CD-ROM, type **mkdir /CDROM**.
  - b. Type **sam &**.
  - c. Select **Disks and File Systems**.
  - d. Select **Disk Devices**.
  - e. Select the entry **CDFS** (for CD-ROM File System), and select **Actions** then **View More Information** from the menu bar.
  - f. Write down the value for Device File, such as **/dev/dsk/c0t2d0**.
  - g. Close the View More Information window.
  - h. Close the Disks and File Systems window.
  - i. Exit sam.
  - j. To mount the CD-ROM device file identified above, type the following command:

```
mount /dev/dsk/c0t2d0 /CDROM
```
  - k. Issue the mount command again and verify that **/CDROM** is listed.

---

## Installing TeamConnection

This section provides step-by-step instructions for the initial installation of TeamConnection, and gives instructions for verifying the installation.

This section is intended for the administrator who does the initial install of TeamConnection for your organization. To perform the installation tasks, you need to be an HP—UX system administrator who can log in as user root and create user accounts. This installation process assumes that you are installing all of the following components on the same workstation:

- TeamConnection family server (including the database)
- Client
- Build server

If you do not follow these instructions as written, the verification step on page 41 might fail.

The time required for installing this program varies depending on your workstation. In general, installation of the product takes about 45 minutes.

After completing the instructions in this section, you will have a family called testfam. Consider this your test family, and use it to verify that TeamConnection is working properly. This family is configured with the default information that IBM ships.

You can use testfam to explore and learn about TeamConnection. Using testfam with the shipped default information will help you determine how to customize TeamConnection to fit your development needs.

**Note:** When installing only a client, follow the instructions in *Getting Started with the TeamConnection Clients*

The following is a list of the tasks you do to install TeamConnection.

Task	Page
Install the TeamConnection components	38
Create a user account for the TeamConnection family	38
Set the environment variables	38
Verify that the ObjectStore server is active	38
Verify installation	39
Start the TeamConnection client	42

## Installing TeamConnection components



**Quick start for experts:** Mount the CD-ROM as **/CDROM**. Use the **"/CDROM/TCCDUTIL.KSH;1"** command to install the TeamConnection components you want.

**Note:** Enter these names and commands exactly as shown, using uppercase letters and quotes, when shown.

**Note:** If you have previously installed CMVC and want to migrate from CMVC to TeamConnection version 2, you need to install TeamConnection version 2 on a separate machine and migrate your CMVC database to TeamConnection. See "Chapter 17. Migrating to TeamConnection version 2" on page 189 for instructions on migrating from CMVC to TeamConnection.

Installing TeamConnection version 2 causes any previous version of all of the components of TeamConnection to be overwritten without any warning messages. You will not be able to go back to the version of TeamConnection that was previously installed without completely reinstalling that version.

If you already have an existing version of TeamConnection installed, ensure that none of the TeamConnection daemons and tools are running.

After you have added and mounted a CD-ROM, do the following to install the TeamConnection components:

1. Log in with root authority.
2. Change the directory to **/CDROM**.
3. Run the installation script by typing **"/CDROM/TCCDUTIL.KSH;1"**.  
In HP-UX, the files on a CD are always shown in uppercase.
4. Follow the instructions in the installation script to install the TeamConnection server.
5. After the installation is complete, type the following commands:  

```
cd ..  
umount /CDROM
```
6. Eject the CD.

## Creating a user account for your TeamConnection family

Each TeamConnection family requires a separate user account.

Log in as root user ID and create a new user account for your family. This can be performed using the HP—UX sam tool.

## Setting the environment variables

Now that TeamConnection for HP-UX is installed, you must set the environment variables for the root user ID. Do the following steps:

1. Add the following variables to the .profile for the root user to start and shut down the ObjectStore database:

```
OS_ROOTDIR=/usr/local/ODI/OS4.0  
export OS_ROOTDIR  
PATH=$PATH:$OS_ROOTDIR/bin:$OS_ROOTDIR/lib  
export PATH  
SHLIB_PATH=$OS_ROOTDIR/lib  
export SHLIB_PATH
```

**Note:** The value of OS\_ROOTDIR must refer to the directory where the ObjectStore code was installed. The default is used in this example.

2. Exit root user ID and log in again as root to update the environment variables.

## Verifying that the ObjectStore server is active

**Note:** If ObjectStore is installed, then the ObjectStore server will be started by the installation script.

Before you create your test family, do the following to verify that the ObjectStore server is active:

1. From a prompt, type:  

```
ps -ef | grep os
```
2. Verify that there is a process called osserver.  
If you have already created a family, then you might see the process oscmgr4 (for the ObjectStore cache manager).

You can start and stop these ObjectStore processes as follows:

- To start the ObjectStore server, type the following command:  
`osserver`
- To shut down the ObjectStore cache manager, type the following command:  
`oscmshtd`
- To shut down the ObjectStore server, type the following command. Substitute the host name of your server for *<hostname>*.  
`ossvrshd -f <hostname>`

## Verifying the installation of TeamConnection

Before you can verify that TeamConnection is installed correctly, you must create your family. When you create your family, you are loading default information shipped by IBM and creating a superuser ID. A superuser ID is required so that at least one person has privileged access to the family to perform special tasks, such as creating other user IDs.

This section leads you through the following procedures to create a test family called *testfam*.

1. Configuring the TeamConnection environment variables in the *.profile*.
2. Verifying that the TeamConnection client is accessible.
3. Creating *testfam*.
4. Starting the family server.
5. Verifying TeamConnection installation.
6. Using the TeamConnection client.

### Step 1 — Configuring TeamConnection environment variables

Environment variables in the *.profile* of the family account need to be configured before you create a TeamConnection family.

It is recommended that the *.profile* for the family be based on the sample *profile.family*. This profile uses all the necessary environment variables to create and use a TeamConnection family.

1. Log in as the user account for the TeamConnection family.
2. Use the following commands to copy the sample *profile.family* as your new *.profile*, substituting the location where TeamConnection is installed for *\$TC\_HOME* and the identifier for the national language version of TeamConnection you are using (such as *enu*) for *language*.

```
mv $HOME/.profile $HOME/.profile.original
cp $TC_HOME/install/language/profile.family $HOME/.profile
```

3. Customize all appropriate entries in the new *.profile*.

Edit the new *.profile*, read the entire file, modify the values as appropriate, and uncomment variables necessary for your operating system.

If you are using RAWFS databases, create a placeholder database in *TC\_DBPATH* using the name *.tcd*. For example, if your *TC\_DBPATH* is */teamc/testfam* and your family name is *testfam*, create a file called */teamc/testfam/testfam.tcd*. It does not matter what is in the file, as long as it exists. The locator file (*/teamc/testfam/testfam.loc* for the previous example) determines the actual database host and location. Also, make sure the environment variable *OS\_DIRMAN\_HOST* is not set.

**Note:** If you are using the CDE environment, edit `.dtprofile` to enable the account to run `.profile` after `.dtprofile`. CDE stands for Common Desktop Environment and is the default for HP—UX 10 as well as many other versions of UNIX. CDE is the standard of the Open Group (c). For instructions, see the header of the sample profile.

4. You will likely want to restrict access to this account. Be sure that your `.profile` (and `.dtprofile` in the CDE environment) have no visibility outside of the user account:

```
chmod u=rw,go= .profile .dtprofile
```

5. Log off and log in as the TeamConnection family to activate the changes to the new profile.

## Step 2 — Ensure that the TeamConnection client is accessible

Because the verification step uses the TeamConnection client, you need to ensure that it is accessible:

1. Type the following TeamConnection command, which does not require connection to a working TeamConnection family.

```
teamc report -testClient
```

2. If the `teamc` command is not found, then ensure that the `TC_HOME` and `PATH` variables are properly set in the `.profile`.
3. If the `teamc` command is found, then verify that the `rop` portion of the output is something like this, assuming that the `LANG` variable is `en_US`:

```
Product Version:                2.0.4
Message catalog language:       English
Environment variables:
Variable:                       Setting:
-----
TC_BECOME                       <your TeamConnection user id>

TC_FAMILY                       <your TeamConnection family name>
```

4. If you see that the entry for "Message catalog language" has the value `English (Internal)`, then the `LANG` or `NLSPATH` variables are not properly set and the TeamConnection client is using its internal messages. It is strongly recommended that the external message catalog be used.

## Step 3 — Creating testfam

To create a test family to be used for installation verification, follow these steps.

1. Ensure that you have logged in with the ID of the family account.
2. Enter the following command from a command prompt:

```
dbcreate
```

This command creates the `testfam` database and an initial TeamConnection user with superuser authority. This user is the family administrator.

After `testfam` is created, you should be able to see the following subdirectories and files in the family account:

### **authorit.Id**

The default authority groups for the family

### **cfgField**

A subdirectory containing default configurable field information

### **comproc.Id**

The default component process for the family



**config** A subdirectory containing default user exit information for the family and security information

**config.ld**

The default configurable field types for the family

**interest.ld**

The default interest groups for the family

**queue** A subdirectory for the messages to be sent by the notify daemon

**relproc.ld**

The default release processes for the family

**schema.adb**

Metadata (definitions of objects such as users, parts, and releases) for the family

**testfam.tcd**

The family database

## Step 4 — Start the family server

To start the testfam family server, type the following command from a command prompt. Use of this command assumes that the path statements and environment variables have been set properly.

```
teamcd testfam
```

See page “Starting the family server” on page 86 for more information about the teamcd command.

## Step 5 — Verifying TeamConnection installation

The following installation verification procedure runs a command file called univunix.ksh, which does the following:

- Ensures that all TeamConnection daemons are stopped (using stopteamc)
- Starts the family daemon and a build agent and processor (using teamcd and temporary TCP/IP socket numbers)
- Runs a suite of TeamConnection actions

Running this command file requires that you have a full build environment set up, including VisualAge C++. “Chapter 20. Installing the build function” on page 267 explains how to install the build function. If you have not installed the build function or VisualAge C++, please omit this procedure.

To verify that TeamConnection installed correctly, type the following at a prompt in the directory where TeamConnection is installed, and then press Enter.

```
univunix
```

For more information on this command, type the following at a command prompt:

```
univunix -help
```

The univunix utility creates a file called univunix.log. Review univunix.log for a completion code of zero on the commands it executes. If all commands execute successfully, then TeamConnection was installed and configured correctly.

When you are finished using the test data, you can restore the original database by issuing the following commands:

```
mv <family>.tcd <family>.tcd.test #or you can delete the test family
mv <family>.db0 <family>.tcd
```

The original database, <family>.db0, is valid only as long as you do not perform any family configuration against the current <family>.tcd. Once you perform any family configuration, please delete <family>.db0.

## Step 6 — Using the TeamConnection client

Before you start the client GUI for the first time, do the following:

1. Copy the sample initial tasks list for the main window. You will need to do this only once:

```
cp $TC_HOME/nls/cfg/$LANG/teamc20.ini $HOME
chmod u+w $HOME/teamc20.ini
```

2. Start the TeamConnection GUI by issuing the following command:

```
teamcgui &
```

You can issue TeamConnection line commands from the family account. To see a list of all users, for example, issue the following command:

```
teamc report -view users -where 1=1
```

## What do you do now?

You now have your initial family installed and running. As mentioned earlier in this chapter, you or someone else in your organization can use this family to verify that TeamConnection is working properly. You can also use this family to explore and learn about TeamConnection, and test configuration changes, user exits, automated backup utilities, and automated build and packaging events.

The remaining chapters of this book provide instructions for installing build servers and instructions for starting and stopping the servers.

To learn more about planning for and creating your TeamConnection production family, see “Part 3. Designing and creating your TeamConnection environment” on page 97 .

---

## Chapter 5. Installing TeamConnection for Solaris

This chapter lists the hardware and software that are required before you can install and use the TeamConnection for Solaris product. It also describes the tasks that you must do before installation, and provides detailed instructions for installing and configuring the TeamConnection family server and client.

### Note:

If you have previously installed CMVC and want to migrate from CMVC to TeamConnection version 3, you need to install TeamConnection version 3 on a separate machine and migrate your CMVC database to TeamConnection. See “Chapter 17. Migrating to TeamConnection version 2” on page 189 for instructions on migrating from CMVC to TeamConnection.

After you install the TeamConnection code in its directory structure (represented by `$TC_HOME`), please keep in mind the following guidelines:

- Do not remove directories, files, or symbolic links from `$TC_HOME`, unless you have uninstalled the product.
- Do not change the file permissions or the ownership of the directories or files in `$TC_HOME`.
- Use `$TC_HOME` only to store the TeamConnection code. Do not use it as the home directory for users and do not use it as the home directory for TeamConnection families.
- If this is your first installation of TeamConnection, accept the default values suggested in the installation instructions and scripts.

---

## Hardware requirements

The following are hardware requirements for the TeamConnection for Solaris **family server** and **build server** machines:

*Table 7. Hardware Requirements for a Solaris TeamConnection Server*

<b>Family server</b>	<ul style="list-style-type: none"><li>• Processor: SPARC or UltraSPARC workstation.</li><li>• Pointing device: A mouse or other pointing device</li><li>• Monitor: Any X11 graphics display supported by the processor</li><li>• Memory: 128 MB memory minimum; more may be needed according to number of users and database size</li><li>• Disk space:<ul style="list-style-type: none"><li>– 500 MB for operating system and prerequisites</li><li>– 200 MB for user data (1+ GB recommended)</li><li>– 200 MB for TeamConnection server code</li><li>– 65 MB for TeamConnection documentation</li><li>– 140 MB for ObjectStore</li><li>– 100 MB for temporary space</li><li>– Amount recommended by operating system for swapper space</li></ul></li><li>• Communications support: Any token-ring or Ethernet local area network (LAN) adapter card that supports TCP/IP and is supported by the workstation</li><li>• A CD-ROM drive (internal or external) for installation of the product</li></ul>
----------------------	---

Table 7. Hardware Requirements for a Solaris TeamConnection Server (continued)

<b>Build server</b>	<ul style="list-style-type: none"><li>• Processor: SPARC or UltraSPARC workstation.</li><li>• Pointing device: A mouse or other pointing device</li><li>• Monitor: Any X11 graphics display supported by the processor</li><li>• Memory: 64 MB memory minimum (128 MB recommended); more may be needed according to the compilers and linkers used</li><li>• Disk space:<ul style="list-style-type: none"><li>– 500 MB for operating system and prerequisites</li><li>– 200 MB for TeamConnection server code</li><li>– 140 MB for ObjectStore</li><li>– 100 MB for temporary space</li><li>– Amount recommended by operating system for swapper space</li></ul></li><li>• Communications support: Any token-ring or Ethernet local area network (LAN) adapter card that supports TCP/IP and is supported by the workstation</li><li>• A CD-ROM drive (internal or external) for installation of the product</li></ul>
---------------------	--

The following are hardware requirements for the TeamConnection for Solaris **client** machine:

Table 8. Hardware Requirements for a Solaris TeamConnection Client

<b>Processor</b>	SPARC and UltraSPARC workstation.
<b>Pointing device</b>	A mouse or other pointing device.
<b>Monitor</b>	Any X11 graphics display supported by the processor.
<b>Memory</b>	64 MB memory minimum.
<b>Disk space</b>	300 MB for operating system and prerequisites, 60 MB for TeamConnection client code, Amount recommended by operating system for swapper space.
<b>Communications support</b>	Any token-ring or Ethernet local area network (LAN) adapter card that supports TCP/IP and is supported by the workstation.
<b>CD-ROM drive</b>	A CD-ROM drive (internal or external) for installation of the product.

---

## Software requirements

The software requirements for the Solaris **family server**, **client**, and **build server** are the following:

Table 9. Software Requirements for TeamConnection for Solaris

<b>Server</b>	Solaris 2.5.1 or a later release of version 2 that includes TCP/IP.
<b>Client</b>	Solaris 2.5.1 or a later release of version 2 that includes TCP/IP.
<b>Build server</b>	Solaris 2.5.1 or a later release of version 2 that includes TCP/IP.

---

## Preparing to install TeamConnection for Solaris

This section explains what you need to do before installing TeamConnection for Solaris. To install TeamConnection, you should be a Solaris system administrator who can log in as user root.

Use the Solaris admintool to create a user account. This tool presents a menu-driven environment for system administration tasks. For more information on admintool, refer to your Solaris operating system documentation.

## Updating TCP/IP files

Before you install the TeamConnection code, you must have the correct version of TCP/IP installed on your workstation. See “Software requirements” on page 44 for the communications software requirements for your operating system.

After TCP/IP is installed, update your TCP/IP services and hosts files.

Do the following steps. In many installations, a name server is used instead of /etc/hosts. Also, many installations distribute /etc/services.

1. Update the services file, which is located in /etc/services.

Include the family name and port address of the TeamConnection server. The port address can be any 4-digit number, as long as it does not already exist in your services file. You might want to ask your TCP/IP administrator to assign you a number.

Type the following entry in your services file:

```
# TeamConnection servers
testfam    ffff/tcp    # port address for the TeamConnection test family
```

**Notes:**

- a. For this installation, replace *ffff* with an appropriate port address.
  - b. Follow the line with a carriage return.
2. Update the TCP/IP hosts file, which is located in /etc/hosts.  
Add the following:
    - IP address.
    - Server name.
    - Alias name of the TeamConnection family server, which is your family name.  
For this initial installation of TeamConnection, the family name is *testfam*.

The following is an example of the entry you would type in your hosts file:

```
9.12.345.67    teamserv.company.com    testfam
```

**Note:** Follow the line with a carriage return. You can use the `hostname` command to get the name of the server.

3. Do the following to verify that the hosts file is specified correctly:

The `tchost.sol` utility is available from the `misc` directory in the CD-ROM for TeamConnection for UNIX.

- Type `tchost.sol family_name`, where *family\_name* is the name of your TeamConnection family. The information returned should match the number and name specified in your hosts file entry. For example, using the entry given in the previous step, the system response would be as follows:  

```
teamserv.company.com is 9.12.345.67
```
- Type `tchost.sol ip_address`, where *ip\_address* is the IP address of your machine. The information returned should match the number and name specified in your hosts file entry. For example, again using the entry given in the previous step, the system response would be as follows:  

```
teamserv.company.com is 9.12.345.67
```

If you do not receive the expected response, contact your TCP/IP administrator to solve the problem.

4. Do the following to verify that you can connect to your TeamConnection family:
  - a. At a prompt, type `ping -s testfam`.
  - b. Press `Ctrl+C` to end the command.

If you receive information that is similar to the following, you can successfully connect to your TeamConnection family:

```
PING teamserv.company.com: 56 data bytes
64 bytes from 1.23.457.78: icmp_seg:0. time=0. ms
64 bytes from 1.23.456.78: icmp_seg:1. time=0. ms
64 bytes from 1.23.456.78: icmp_seg:2. time=0. ms
```

If you receive the message `ping: unknown host testfam`, you cannot connect to the family. Verify that the data you entered in the `hosts` and `services` files is correct, and then try the command again. If you still do not get the correct response, contact your TCP/IP administrator to solve the problem.

## Preparing the syslog file

TeamConnection and database errors are recorded in the `syslog` file. The information recorded in this file will help you resolve problems when they occur.

Before TeamConnection can record these errors, you must activate the `syslog` daemon. Do the following to activate the `syslog` daemon:

1. The `syslog` file is located in `/var/adm/messages`. If the log file does not exist, type the following `touch` command to create it. When you create the log file, follow the directions for setting permissions for your operating system.

```
touch /var/adm/messages
chmod 666 /var/adm/messages
chown root /var/adm/messages
chgrp other /var/adm/messages
```

If you don't want to use `chmod 666`, you can use `chmod a=rw /var/adm/messages` or `chmod ugo=rw /var/adm/messages`.

The command `chown root:other` is a short way to do the following:

```
chown root /var/adm/messages
chgrp other /var/adm/messages
```

2. Add the following lines to the `/etc/syslog.conf` file:

```
*.warning /var/adm/messages
```
3. Type the following to stop and then restart the `syslog` daemon:

```
ps -ef | grep syslogd
# Find the process id for syslogd
kill -9 # Restart syslogd
/usr/sbin/syslogd
```

4. Type the following command to verify that the `syslog` daemon is running:

```
ps -ef | grep syslog
```

If the daemon is running, you will see one process for `syslogd`.

5. Do the following to verify that the `syslog` daemon is able to write to the `syslog` file:
  - a. Log in as another user ID.

- b. Type `su root` and an incorrect password to add an error message to the syslog.

To quickly look at the last ten lines of the syslog, type the following command, substituting the proper name of your syslog file for *syslogFileName*:

```
tail syslogFileName
```

If the syslog file is configured properly, it will contain a message similar to the following:

```
Oct 14 16:17:13 oem-sn10 su: 'su root' failed for buildtc on /dev/pts/4
```

6. To clean up the syslog, type the following command, substituting the proper name of your syslog file for *syslogFileName*:

```
cp /dev/null syslogFileName
```

7. Monitor the syslog file at regular intervals so that you can perform any maintenance or problem resolution.

## Installing TeamConnection

This section provides step-by-step instructions for the initial installation of TeamConnection, and gives instructions for verifying the installation.

This section is intended for the administrator who does the initial installation of TeamConnection for your organization. To perform the installation tasks, you need to be a Solaris system administrator who can log in as user root and create user accounts. This installation process assumes that you are installing all of the following components on the same workstation:

- TeamConnection family server (including ObjectStore database)
- Client
- Build server

If you do not follow these instructions as written, the verification step on page “Step 5 — Verify TeamConnection installation” on page 51 might fail.

The time required for installing this program varies depending on your workstation. In general, installation of the product takes about 45 minutes.

After completing the installation and verification instructions in this section, you will have a family called testfam. Consider this your test family, and use it to verify that TeamConnection is working properly. This family is configured with the default information that IBM ships.

You can use testfam to explore and learn about TeamConnection. Using testfam with the shipped default information will help you determine how to customize TeamConnection to fit your development needs.

**Note:** When installing only a client, follow the instructions in *Getting Started with the TeamConnection Clients*

The following is a list of the tasks you do to install TeamConnection:

Task	Page
Install the TeamConnection components	48
Create a user account for the TeamConnection family	48

Task	Page
Set the environment variables	49
Verify that the ObjectStore server is active	49
Verify the installation	"Verifying the installation of TeamConnection" on page 49
Start the TeamConnection client	"Step 6 — Using the TeamConnection client" on page 52

## Installing TeamConnection components



**Quick start for experts:** Insert the CD-ROM and change directory to `/cdrom/teamcXXX` (where XXX is the version number, such as teamc300). Use the `tcinst.ksh` command to install the TeamConnection components you want.

**Note:** If you have previously installed CMVC and want to migrate from CMVC to TeamConnection version 3, you need to install TeamConnection version 3 on a separate machine and migrate your CMVC database to TeamConnection. See "Chapter 17. Migrating to TeamConnection version 2" on page 189 for instructions on migrating from CMVC to TeamConnection.

After you have inserted the CD-ROM, do the following to install the TeamConnection components:

1. Log in with root authority.
2. Change the directory to `/cdrom/teamcXXX` (where XXX is the version number, such as teamc300).
3. Run the installation script `tcinst.ksh` by typing `./tcinst.ksh`.  
In Solaris, the files on a CD-ROM are always shown in lowercase, and they need to be specified in lowercase.
4. Follow the instructions in the installation script to install the TeamConnection server and client.
5. After the installation is complete, type the following commands:  
`cd ..`
6. Eject the CD:  
`eject`

## Creating a user account for your TeamConnection family

Each TeamConnection family requires a separate user account.

Log in as root user ID and create a new user account for your family. This can be performed using the Solaris SMIT tool.



## Setting the environment variables for ObjectStore

Now that TeamConnection for Solaris is installed, you must set the environment variables for the root user ID. These variables are needed to manage the database. Do the following steps:

1. Add the following variables to the .profile for the root user to start and shut down the ObjectStore database:

```
OS_ROOTDIR=/opt/ODI/OS5.0/cset
export OS_ROOTDIR
PATH=$PATH:$OS_ROOTDIR/bin:$OS_ROOTDIR/lib
export PATH
LIBPATH=$OS_ROOTDIR/lib
export LIBPATH
```

2. Exit the root user ID and log in again as root to update the environment variables.

## Verifying that the ObjectStore server is active

**Note:** If ObjectStore is installed, then the ObjectStore server will be started by the installation script.

Before you create your test family, do the following to verify that the ObjectStore server is active:

1. From a prompt, type:  

```
ps -ef | grep os
```
2. Verify that there is a process called ossserver.  
If you have already created a family, then you might see the process oscmgr4 (for the ObjectStore cache manager).

You can start and stop these ObjectStore processes as follows:

- To start the ObjectStore server, type the following command:  

```
osserver
```
- To shut down the ObjectStore cache manager, type the following command:  

```
oscmshtd
```
- To shut down the ObjectStore server, type the following command. Substitute the host name of your server for *<hostname>*.  

```
ossvrshd -f <hostname>
```

## Verifying the installation of TeamConnection

Before you can verify that TeamConnection is installed correctly, you must create your family. When you create your family, you are loading default information shipped by IBM and creating a superuser ID. A superuser ID is required so that at least one person has privileged access to the family to perform special tasks, such as creating other user IDs.

This section leads you through the following procedures to create a test family called *testfam*.

1. Configuring the TeamConnection environment variables in the .profile.
2. Verifying that the TeamConnection client is accessible.
3. Creating testfam.
4. Starting the family server.
5. Verifying TeamConnection installation.

6. Using the TeamConnection client.

**Step 1 — Configuring TeamConnection environment variables:** Environment variables in the .profile of the family account need to be configured before you create a TeamConnection family.

It is recommended that the .profile for the family be based on the sample profile.family. This profile uses all the necessary environment variables to create and use a TeamConnection family.

1. Log in as the user account for the TeamConnection family.
2. Use the following commands to copy the sample profile.family as your new .profile, substituting the location where TeamConnection is installed for `$TC_HOME` and the identifier for the national language version of TeamConnection you are using (such as `en_us`) for `$LANG`.

```
mv $HOME/.profile $HOME/.profile.original
cp $TC_HOME/install/$LANG/profile.family $HOME/.profile
```

3. Customize all appropriate entries in the new .profile.

Edit the new .profile, read the entire file, modify the values as appropriate, and uncomment variables necessary for your operating system.

If you are using RAWFS databases, create a placeholder database in `TC_DBPATH` using the name `.tcd`. For example, if your `TC_DBPATH` is `/teamc/testfam` and your family name is `testfam`, create a file called `/teamc/testfam/testfam.tcd`. It does not matter what is in the file, as long as it exists. The locator file (`/teamc/testfam/testfam.loc` for the previous example) determines the actual database host and location. Also, make sure the environment variable `OS_DIRMAN_HOST` is not set.

**Note:** If you are using the CDE environment, edit `.dtprofile` to enable the account to run `.profile` after `.dtprofile`. CDE stands for Common Desktop Environment and is the default for Solaris as well as many other versions of UNIX. CDE is the standard of the Open Group (c). For instructions, see the header of the sample profile.

4. You will likely want to restrict access to this account. Be sure that your .profile (and .dtprofile in the CDE environment) have no visibility outside of the user account. To restrict access to the account, issue the following command:

```
chmod u=rw,go= .profile .dtprofile
```

5. Log off and log in as the TeamConnection family to activate the changes to the new profile.

**Step 2 — Ensure that the TeamConnection client is accessible:** Because the verification step uses the TeamConnection client, you need to ensure that it is accessible:

1. Type the following TeamConnection command, which does not require connection to a working TeamConnection family.

```
teamc report -testClient
```

2. If the `teamc` command is not found, then ensure that the `TC_HOME` and `PATH` variables are properly set in the .profile.
3. If the `teamc` command is found, then verify that the `rop` portion of the output is something like this, assuming that the `LANG` variable is `en_US`:

```
Product Version:          3.0.0
Message catalog language: English
Environment variables:
Variable:                 Setting:
-----
```

```
TC_BECOME          <your TeamConnection user id>

TC_FAMILY          <your TeamConnection family name>
```

4. If you see that the entry for "Message catalog language" has the value English (Internal), then the LANG or NLSPATH variables are not properly set and the TeamConnection client is using its internal messages. It is strongly recommended that the external message catalog be used.

**Step 3 — Creating testfam:** To create a test family to be used for installation verification, follow these steps.

1. Ensure that you have logged in with the ID of the family account.
2. Enter the following command from a command prompt:

```
dbcreate
```

This command creates the testfam database and an initial TeamConnection user with superuser authority. This user is the family administrator.

After testfam is created, you should be able to see the following subdirectories and files in the family account:

**authorit.ld**

The default authority groups for the family

**cfgField**

A subdirectory containing default configurable field information

**comproc.ld**

The default component process for the family

**config** A subdirectory containing default user exit information for the family and security information

**config.ld**

The default configurable field types for the family

**interest.ld**

The default interest groups for the family

**queue** A subdirectory for the messages to be sent by the notify daemon

**relproc.ld**

The default release processes for the family

**schema.adb**

Metadata (definitions of objects such as users, parts, and releases) for the family

**testfam.tcd**

The family database

**Step 4 — Start the family server:** To start the testfam family server, type the following command from a command prompt. Use of this command assumes that the path statements and environment variables have been set properly.

```
teamcd testfam
```

**Step 5 — Verify TeamConnection installation:** The following installation verification procedure runs a command file called univunix, which does the following:

- Ensures that all TeamConnection daemons are stopped (using stopteamc)
- Starts the family daemon and a build agent and processor (using teamcd and temporary TCP/IP socket numbers)

- Runs a suite of TeamConnection actions

Running this command file requires that you have a full build environment set up, including VisualAge C++. “Chapter 20. Installing the build function” on page 267 explains how to install the build function. If you have not installed the build function or VisualAge C++, please omit this procedure.

To verify that TeamConnection installed correctly, type the following at a prompt in the directory where TeamConnection is installed, and then press Enter.

```
univunix
```

For more information on this command, type the following at a command prompt:

```
univunix -help
```

The univunix utility creates a file called univunix.log. Review univunix.log for a completion code of zero on the commands it runs. If all commands run successfully, then TeamConnection was installed and configured correctly.

When you are finished using the test data, you can restore the original database by issuing the following commands:

```
mv <family>.tcd <family>.tcd.test #or you can delete the test family
mv <family>.db0 <family>.tcd
```

The original database, <family>.db0, is valid only as long as you do not perform any family configuration against the current <family>.tcd. Once you perform any family configuration, please delete <family>.db0.

**Step 6 — Using the TeamConnection client:** Before you start the client GUI for the first time, do the following:

1. Copy the sample initial tasks list for the main window. You will need to do this only once:

```
cp $TC_HOME/nls/cfg/$LANG/teamc20.ini $HOME
chmod u+w $HOME/teamc20.ini
cp $TC_HOME/install/$LANG/Teamcgui.user $HOME/Teamcgui
chmod u+w $HOME/Teamcgui
```

Where \$TC\_HOME is the location where the TeamConnection code was installed.

2. Start the TeamConnection GUI by issuing the following command:

```
teamcgui &
```

You can issue TeamConnection line commands from the family account. To see a list of all users, for example, issue the following command:

```
teamc report -view users -where 1=1
```

## What do you do now?

You now have your initial family installed and running. As mentioned earlier in this chapter, you or someone else in your organization can use this family to verify that TeamConnection is working properly. You can also use this family to explore and learn about TeamConnection, and test configuration changes, user exits, automated backup utilities, and automated build and packaging events.

The remaining chapters of this book provide instructions for installing build servers and instructions for starting and stopping the servers.

To learn more about planning for and creating your TeamConnection production family, see “Part 3. Designing and creating your TeamConnection environment” on page 97 .



---

## Chapter 6. Installing TeamConnection for OS/2

This chapter lists the hardware and software that are required before you can install and use the TeamConnection for OS/2 product. It also describes the tasks that you must do before installation, and provides detailed instructions for installing and configuring the TeamConnection family server and client.

### Note:

If you have installed a previous version of TeamConnection, you cannot update your old files with version 3. Instead, you need to install version 3 on a separate machine and migrate your database to version 3. See "Chapter 17. Migrating to TeamConnection version 2" on page 189 for instructions on migrating to version 3.

After you install the TeamConnection code in its directory structure (represented by x:\teamc), please keep in mind the following guidelines:

- Do not remove directories, files, or symbolic links from x:\teamc, unless you have uninstalled the product.
- Do not change the file permissions or the ownership of the directories or files in x:\teamc.
- Use x:\teamc only to store the TeamConnection code. Do not use it as the home directory for users and do not use it as the home directory for TeamConnection families.
- If this is your first installation of TeamConnection, accept the default values suggested in the installation instructions and scripts.

---

## Hardware requirements

TeamConnection runs on IBM and IBM-compatible personal computer hardware configurations supported by IBM OS/2 Warp or later.

**Note:** Required memory and DASD can vary depending on your installation and configuration choices and user application requirements.

The following are hardware requirements for the TeamConnection OS/2 **family server** and **build server** machines:

Table 10. Hardware Requirements for an OS/2 TeamConnection Server

<b>Family server</b>	<ul style="list-style-type: none"> <li>• Processor: 90 MHz Pentium-based processor or higher</li> <li>• Pointing device: A mouse or other pointing device</li> <li>• Monitor: VGA or higher resolution with the appropriate adapter</li> <li>• Memory: 48 MB memory minimum; more may be needed according to number of users and database size</li> <li>• Disk space: <ul style="list-style-type: none"> <li>– 200 MB for operating system and prerequisites</li> <li>– 200 MB for user data (1+ GB recommended)</li> <li>– 60 MB for TeamConnection server code</li> <li>– 140 MB for ObjectStore</li> <li>– 128 MB for swapper space (150+ MB recommended)</li> </ul> </li> <li>• Communications support: Network card supported by TCP/IP for OS/2</li> <li>• A CD-ROM drive (internal or external) for installation of the product</li> </ul>
<b>Build server</b>	<ul style="list-style-type: none"> <li>• Processor: 66 MHz 486-based processor or higher</li> <li>• Pointing device: A mouse or other pointing device</li> <li>• Monitor: VGA or higher resolution with the appropriate adapter</li> <li>• Memory: 32 MB memory minimum; more may be needed according to compilers and linkers used</li> <li>• Disk space: <ul style="list-style-type: none"> <li>– 75 MB for operating system and prerequisites</li> <li>– 15 MB for TeamConnection build server code</li> <li>– 140 MB for ObjectStore</li> <li>– 45 MB for swapper space</li> </ul> </li> <li>• Communications support: Network card supported by TCP/IP for OS/2</li> <li>• A CD-ROM drive (internal or external) for installation of the product</li> </ul>

The following are hardware requirements for the TeamConnection OS/2 **client** machine:

Table 11. Hardware Requirements for an OS/2 TeamConnection Client

<b>Processor</b>	66 MHz 486-based processor or higher
<b>Pointing device</b>	A mouse or other pointing device
<b>Monitor</b>	VGA or higher resolution with the appropriate adapter
<b>Memory</b>	12 MB minimum
<b>Disk space</b>	<ul style="list-style-type: none"> <li>• 75 MB for operating system and prerequisites</li> <li>• 25 MB for TeamConnection client code</li> <li>• 32 MB for swapper space</li> </ul>
<b>Communications support</b>	Network card supported by TCP/IP for OS/2
<b>CD-ROM drive</b>	A CD-ROM drive (internal or external) for installation of the product



---

## Software requirements

The following are software requirements for TeamConnection for OS/2:

*Table 12. Software Requirements for TeamConnection for OS/2*

<b>Server</b>	One of the following: <ul style="list-style-type: none"><li>• OS/2 Warp Version 3 (part number 83G8100) or Version 4 (84H1428) and one of the following:<ul style="list-style-type: none"><li>– TCP/IP Version 3.0 for OS/2 Warp (part number 33H9749)</li><li>– Anynet/2 Version 2.0 (part number 87G7776)</li></ul></li><li>• OS/2 Warp Connect Version 3 (part number 10H9800)</li><li>• OS/2 Warp Server Version 4 (part number 25H8002)</li><li>• OS/2 Warp Server Advanced Version 4 (part number 25H8030)</li><li>• OS/2 Warp Server Advanced Version 4 SMP Feature (part number 28H0150)</li></ul>
<b>Client</b>	One of the following: <ul style="list-style-type: none"><li>• OS/2 Warp Version 3 (part number 83G8100) and one of the following:<ul style="list-style-type: none"><li>– TCP/IP Version 3.0 for OS/2 Warp (part number 33H9749)</li><li>– Anynet/2 Version 2.0 (part number 87G7776)</li></ul></li><li>• OS/2 Warp Version 4 (part number 84H1426) and one of the following:<ul style="list-style-type: none"><li>– TCP/IP Version 3.0 for OS/2 Warp (part number 33H9749)</li><li>– Anynet/2 Version 2.0 (part number 87G7776)</li></ul></li><li>• OS/2 Warp Connect Version 3 (part number 10H9800)</li><li>• OS/2 Warp Server Version 4 (part number 25H8002)</li><li>• OS/2 Warp Server Advanced Version 4 (part number 25H8030)</li><li>• OS/2 Warp Server Advanced Version 4 SMP Feature (part number 28H0150)</li></ul> <p><b>Note:</b> If double-byte character set (DBCS) support is required, one of the following DBCS versions of OS/2:</p> <ul style="list-style-type: none"><li>– IBM OS/2 J version 2.11 or later</li><li>– IBM OS/2 H version 2.11 or later</li><li>– IBM OS/2 T version 2.11 or later</li><li>– IBM OS/2 P version 2.11 or later</li></ul>
<b>Build server</b>	One of the following: <ul style="list-style-type: none"><li>• OS/2 Warp Version 3 (part number 83G8100) and one of the following:<ul style="list-style-type: none"><li>– TCP/IP Version 3.0 for OS/2 Warp (part number 33H9749)</li><li>– Anynet/2 Version 2.0 (part number 87G7776)</li></ul></li><li>• OS/2 Warp Connect Version 3 (part number 10H9800)</li><li>• OS/2 Warp Server Version 4 (part number 25H8002)</li><li>• OS/2 Warp Server Advanced Version 4 (part number 25H8030)</li><li>• OS/2 Warp Server Advanced Version 4 SMP Feature (part number 28H0150)</li></ul>
<b>Distribution function</b>	NetView Distribution Manager/2 Version 2.0 or later (53G3924)

---

## Preparing to install TeamConnection for OS/2



If you use SmartGuides to install TeamConnection, then your TCP/IP services and hosts files are set up for you and you can skip this section.

Before you install the TeamConnection code, you must have the correct version of TCP/IP installed on your workstation. See “Software requirements” on page 57 for the communications software requirements for your operating system.

After TCP/IP is installed, update your TCP/IP services and hosts files.

Do the following steps.

1. Update the services file, which is located in the directory where TCP/IP is installed. To determine the directory name, type `echo %etc%` at a prompt. Include the family name and port address of the TeamConnection server. The port address can be any 4-digit number, as long as it does not already exist in your services file. You might want to ask your TCP/IP administrator to assign you a number.

Type the following entry in your services file:

```
# TeamConnection servers
testfam    ffff/tcp    # port address for the TeamConnection test family
```

**Notes:**

- a. For this installation, replace *ffff* with the appropriate port address.
  - b. Follow the line with a carriage return.
2. Update the TCP/IP hosts file. If you have a hosts file, it is located in the directory where TCP/IP is installed. To determine the directory name, type `echo %etc%` at a prompt. If the hosts file does not exist, you must configure it through TCP/IP.

Add the following:

- IP address.
- Server name.
- Alias name of the TeamConnection family server, which is your family name. For this initial installation of TeamConnection, the family name is *testfam*.
- Alias name for the *build socket*. For this initial installation of TeamConnection, use *bldsock*.

The following is an example of the entry you would type in your hosts file:

```
9.12.345.67    teamserv.company.com    testfam    bldsock
```

**Note:** Follow the line with a carriage return. You can use the `hostname` command to get the name of the server.

3. Do the following to verify that the hosts file is specified correctly:
  - Type `host family_name`, where *family\_name* is the name of your TeamConnection family. The information returned should match the number and name specified in your hosts file entry. For example, using the entry given in the previous step, the system response would be as follows:

```
teamserv.company.com = 9.12.345.67
```

- Type host *ip\_address*, where *ip\_address* is the IP address of your machine. The information returned should match the number and name specified in your hosts file entry. For example, again using the entry given in the previous step, the system response would be as follows:  
9.12.345.67 = teamserv.company.com

If you do not receive the expected response, contact your TCP/IP administrator to solve the problem.

4. Do the following to verify that you can connect to your TeamConnection family:
  - a. At a prompt, type ping testfam.
  - b. Press Ctrl+C to end the command.

If you receive information that is similar to the following, you can successfully connect to your TeamConnection family:

```
PING teamserv.company.com: 56 data bytes
64 bytes from 1.23.457.78: icmp_seg:0. time=0. ms
64 bytes from 1.23.456.78: icmp_seg:1. time=0. ms
64 bytes from 1.23.456.78: icmp_seg:2. time=0. ms
```

If you receive the message unknown host testfam, you cannot connect to the family. Verify that the data you entered in the hosts and services files is correct, and then try the command again. If you still do not get the correct response, contact your TCP/IP administrator to solve the problem.

5. Do the following to verify that you can connect to your TeamConnection build server:
  - a. At a prompt, type ping bldsock.
  - b. Press Ctrl+C to end the command.

If you receive information that is similar to the following, you can successfully connect to your TeamConnection family:

```
PING teamserv.company.com: 56 data bytes
64 bytes from 1.23.457.78: icmp_seg:0. time=0. ms
64 bytes from 1.23.456.78: icmp_seg:1. time=0. ms
64 bytes from 1.23.456.78: icmp_seg:2. time=0. ms
```

If you receive the message unknown host bldsock, you cannot connect to the build server. Verify that the data you entered in the hosts and services files is correct, and then try the command again. If you still do not get the correct response, contact your TCP/IP administrator to solve the problem.

Do not install the TeamConnection components until the ping commands successfully complete.

---

## Installing TeamConnection using SmartGuides

This section describes the steps to follow when installing the TeamConnection components using the TeamConnection SmartGuide for installation. The SmartGuide is a wizard that automates many of the installation and configuration steps for you. SmartGuides ask you for information about your system and then use the information you provide to set up your server. The TeamConnection SmartGuide for server installation can perform the following installation steps for you:

1. Install the TeamConnection server code.
2. Create a test family.

3. Verify that TeamConnection was installed properly.

If you choose not to use the SmartGuide for installation, you will need to manually install and configure your server as described in “Installing and configuring TeamConnection manually” on page 61. It is assumed that you have already completed the pre-installation tasks noted in “Preparing to install TeamConnection for OS/2” on page 58.

This section is intended for the administrator who does the initial install of TeamConnection for your organization. It assumes that you are installing all of the following components on the same workstation:

- TeamConnection family server (including ObjectStore database)
- Client
- Build server
- MVS build server
- CMVC migration utilities
- Online documentation

This procedure leads you through the installation process

**Note:** If you have installed a previous version of TeamConnection, you cannot update your files with version 3. Instead, you need to install version 3 on a separate machine and migrate your database to version 3. See “Chapter 17. Migrating to TeamConnection version 2” on page 189 for instructions on migrating to version 3.

The time required for installing this program varies depending on your workstation. In general, installation of the product takes about 15 minutes.

After completing the instructions in this section, you will have a family called testfam. Consider this your test family, and use it to verify that TeamConnection is working properly. This family is configured with the default information that IBM ships. You can use testfam to explore and learn about TeamConnection. Using testfam with the shipped default information will help you determine how to customize TeamConnection to fit your development needs.

**Note:** When installing only a client, follow the instructions in *Getting Started with the TeamConnection Clients*

To install TeamConnection for OS/2, follow these steps:

1. Insert the CD-ROM into the reader.
2. From a command prompt, type `x:\install`, where *x* is the CD-ROM drive letter. The TeamConnection SmartGuide Menu appears.
3. Select **SmartGuide for Server Installation**. The TeamConnection Server Installation SmartGuide welcome window appears.  
If you do not want to use the SmartGuide, select **Cancel**.
4. Read each SmartGuide panel carefully, supply any requested information, and follow the instructions.

During setup, you will be asked to shutdown and restart your computer. The SmartGuide will resume its setup functions after your computer is started.

During the installation verification process, you will be asked to provide information about your test build server, such as the socket name (bldsock) you provided for it in your TCP/IP services file.

---

## Installing and configuring TeamConnection manually

This section is intended for the administrator who does the initial install of TeamConnection for your organization. It assumes that you are installing all of the following components on the same workstation:

- TeamConnection family server (including the database)
- Client
- Build server
- MVS build server
- CMVC migration utilities
- Online documentation

This section provides step-by-step instructions for the initial installation of TeamConnection, and gives instructions for verifying the installation. If you do not follow these instructions as written, the verification step on page “Step 3. Verify TeamConnection installation” on page 67 might fail.

## Installing TeamConnection

To install TeamConnection, follow these steps:

### Step 1: Starting the installation

To install TeamConnection for OS/2, follow these steps:

1. Insert the CD-ROM into the reader.
2. From a command prompt, type `x:\install`, where *x* is the CD-ROM drive letter. The TeamConnection SmartGuide Menu appears.
3. Select **OS/2 Software Installer Program** to exit the SmartGuide and install TeamConnection manually using Software Installer.
4. From the Software Installer window, select **Continue** to begin installation.

### Step 2: Selecting installation options

1. From the Install window, indicate that you want to have your config.sys file updated. This is the default. If you want existing installation files overwritten during the installation, select **Overwrite files**.  
If you select not to have your config.sys file updated automatically, you will need to update the file manually after the installation program completes. TeamConnection creates a file called config.add that contains your entire config.sys file with environment variables and changes to the PATH, LIBPATH, and DPATH statements.
2. Select **OK** or press Enter. The Install - directories window appears.

### Step 3: Selecting the components for installation

Use the Install - directories window to select which components you want to install and where you want them installed.

1. Select **Select all** to install all the TeamConnection components.

2. To verify that the default drive has sufficient space, or to see what other drives are available, select **Disk space**.

If you want to use a different drive, do the following from the Disk space window:

- a. Select the drive that you want to use.
  - b. Select **OK** or press Enter. The Install - directories window appears.
3. Select **Install** from the Install - directories window to start the installation. The Install - progress window shows you the progress of the installation.

#### Step 4: Completing the installation

1. After the installation completes, the Installation and Maintenance window appears with an indication that TeamConnection successfully installed. Select **OK** or press Enter to exit the TeamConnection Installation and Maintenance window.
  2. Verify that the installation program created a folder called **TeamConnection Group**, which includes several program icons.
  3. If TeamConnection automatically updated your config.sys file during installation, restart your system now. Go to "Configuring and starting the database server" on page 63 to continue with the installation process.
- If you need to manually add the environment variables to your config.sys file, go to "Setting the environment variables".

## Setting the environment variables

If you did not select to have the installation program update your config.sys file with the TeamConnection environment variables, TeamConnection created a file called config.add. Update your config.sys file with these environment variables.

We recommend that you use the default values for this initial installation. If you change the values, you will need to provide them when you set up the TeamConnection server.

```
LIBPATH=x:\teamcInstallPath\DLL;x:\teamcInstallPath\NLS\MSG\zzz
```

```
PATH=x:\teamcInstallPath\BIN;x:\teamcInstallPath;x:\teamcInstallPath\NLS\MSG\zzz
```

```
DPATH=x:\teamcInstallPath;x:\teamcInstallPath\NLS\MSG\zzz
```

```
NLSPATH=x:\teamcInstallPath\NLS\MSG\zzz\%N
```

```
TC_BECOME=user
```

```
TC_DBPATH=x:\teamcInstallPath\testfam
```

```
TC_FAMILY=testfam
```

```
TC_USER=user
```

```
TMP=x:\teamcInstallPath\temp
```

```
HELP=x:\teamcInstallPath\NLS\MSG\zzz
```

```
LOCPATH=x:\teamcInstallPath\LOCALE
```

```
OS_TMPDIR=x:\teamcInstallPath\TEMP
```

```
TC_WWWPATH=x:\teamcInstallPath\WWW
```

**Note:** In these variable settings, *x:\teamcInstallPath* is the drive and directory path in which TeamConnection is installed, *zzz* is the language subdirectory, such as *enu* for US English, and *user* is the USER environment variable you specified when TCP/IP was installed. If the TMP environment variable was

previously set by another application, TeamConnection uses the current value. Make sure OS\_TMPDIR points to an existing directory.

## Configuring and starting the database server

The TeamConnection server communicates with the ObjectStore database through an ObjectStore server. This database was installed with the family server.

**Note:** You can have only one version of the ObjectStore database on the server machine.

Do the following to configure the ObjectStore server:

1. Start the ObjectStore for OS/2 Setup program to initialize the ObjectStore server parameters. Start the program in one of two ways:
  - From the TeamConnection Group folder on the desktop, select the **ObjectStore Setup Version 4.0** icon.
  - From the directory where the ObjectStore server is installed (teamc\bin is the default), type ossetup and press Enter.

The ObjectStore for OS/2 Setup window appears.

2. Select **Auto start server** if it is not already selected.
3. Select **Initialize Server**. This push button is named **Reinitialize Server** if you are re-installing ObjectStore. The Server Initialization window appears.  
If ObjectStore does not initialize properly, verify that the OS\_NETWORK environment variable in your config.sys file correctly reflects your client's network configuration.
4. Change the default log file's name, drive, or directory if appropriate.
5. Select **OK**.
6. If you are installing for the first time, the window Initializing Server for File Databases appears. Select **Initialize Server**. If you are re-installing ObjectStore, the Confirm window appears. Select **Yes**.
7. The ObjectStore for OS/2 Setup window appears. Select **Exit**.
8. Start the ObjectStore server in one of two ways:
  - From the TeamConnection Group folder on the desktop, select the **ObjectStore Server** icon.
  - From the directory where the ObjectStore server is installed (teamc\bin is the default), type the following command and press Enter.  

```
start /min osserver
```

Verify that the ObjectStore server (OSSERVER) is active in the Window List. To do this, press Ctrl+Esc.

If you experience problems configuring and starting the ObjectStore server, the following files can provide you additional information about ObjectStore errors. These files are located in the teamc\temp directory.

- osserver.txt
- osc4.out
- oss.out



## Verifying the installation of TeamConnection

Do the following to verify that TeamConnection installed successfully.

### Step 1. Create your test family

Before you can verify that TeamConnection is installed correctly, you must create your family. When you create your family, you are loading default information shipped by IBM and creating a superuser ID. A superuser ID is required so that at least one person has privileged access to the family to perform special tasks, such as creating other user IDs.

This section leads you through the following procedures to create a test family called *testfam*.

1. Verifying that the database server is active
2. Creating testfam

**Verifying that the database server is active:** Before you create your test family, verify that the database server is active in the Window List.

**Creating testfam:** To create a test family to be used for installation verification, follow these steps.

To start the family administrator program, select the TeamConnection Family Administrator from the TeamConnection group.

1. From the TeamConnection Family Administrator window, select **Family → New → Default**.

A settings notebook opens. You use this window to set up your test family. This notebook contains several pages, but for now you need only the **Family Information** and **Initial Superuser** pages. See “Part 3. Designing and creating your TeamConnection environment” on page 97 for more information on all of the family options you can set using this notebook.

**Note:** In these instructions, the values you can enter appear as follows. TeamConnection’s installation verification procedure requires that some of these values be set exactly as shown.

- **Boldface** indicates values you need to enter exactly as shown.
- *Italics* indicates values that you can substitute to suit your needs or local conventions.

2. Complete the fields on the **Family Information** page of the settings notebook as follows:

**Name** **testfam**

**Path** *d:\dbpath.*

Specify the fully-qualified path name of the directory where you want testfam stored. Make sure this directory does not already exist.

TeamConnection creates testfam in a subdirectory of the path you specify. This subdirectory has the same name as the family. If you specify **c:\proddev** as the path name for testfam, for example, TeamConnection places all files related to the family in the directory path c:\proddev\testfam.

**Note:** If a testfam directory already exists, you will need to delete it before you proceed. This procedure will fail if a testfam directory already exists.



**Port** *xxxx*

Specify the TCP/IP port address for testfam, which was set in your TCP/IP services file before you installed the product.

**Mailer** *mailexit*

3. Complete the fields on the **Initial Superuser** page of the settings notebook as follows:

**Login** *userID*

Specify a user ID for the superuser for the family. For single-user platforms (OS/2), use the value set for the TC\_USER environment variable. To see this value, type the following from a command prompt and look for the TC\_USER variable:

```
set | more
```

For multiuser platforms (AIX, HP-UX, Solaris, Windows NT) specify the user's system login ID.

**Name** *userName*

Specify the full name of the superuser, for example, Thomas C. Jones.

**Host** *hostname*

Specify the TCP/IP host name for the family server machine, which was set in your TCP/IP hosts file before you installed the product.

To see this value, type the following from a command prompt:

```
hostname
```

**Note:** You do not need to use the fully-qualified host name. You can, for example, specify *myServer* instead of *myServer.myCompany.com*.

**User** Specify the TeamConnection user ID for the superuser. If you omit this parameter, it defaults to the value specified in the **Login** field (for multi-user platforms) or to the value of the TC\_USER environment variable (for single-user platforms). It is a good idea to give the superuser an ID that is readily identifiable as a superuser. A good way to do this is to preface the user ID with *su\_*, such as *su\_john*.

**Password**

If you want to use password security, you must specify the password to be used to verify the superuser's access to the TeamConnection server. If you do not specify the password for superuser access, then no one will be able to access the database. To use password security, you need to set the **Security level** field on the **Security** page of the family settings notebook to **PASSWORD\_ONLY** or **PASSWORD\_OR\_HOST**.

4. After you have set all the values on the **Family Information** and **Initial Superuser** pages of the family settings notebook, select the **Create** push button.

As TeamConnection creates testfam, the **Create Family** window appears showing the commands executed and the status of each step in the family creation process. The following is an example of this window:

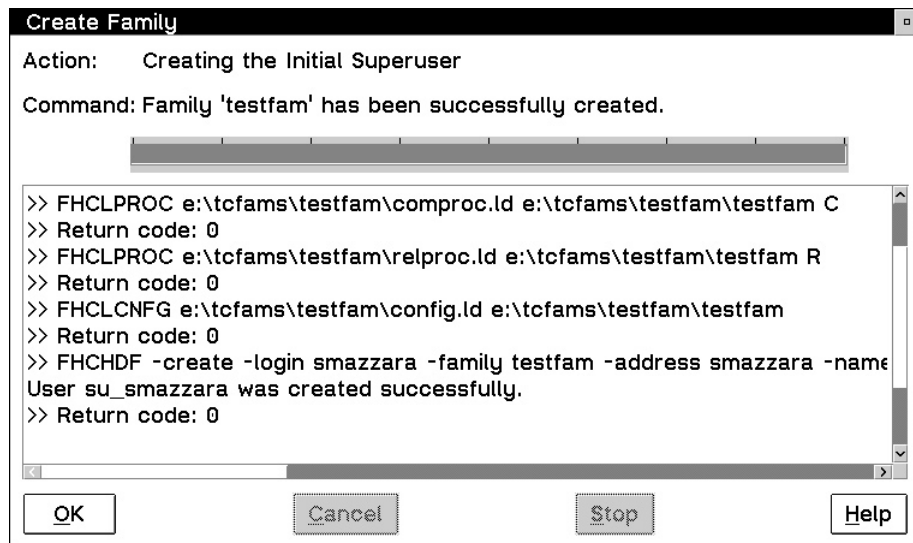


Figure 4. Create Family window

For an explanation of the commands executed in each step of the family creation process, see “Appendix A. Family administration commands” on page 369 .

After testfam is created, you should be able to see the following:

- A family administrator icon labeled testfam in the **TeamConnection Family Administrator** window.
- A subdirectory called testfam in the directory path that you specified on the family settings notebook.

This subdirectory, in turn, contains the following: Copy or backup your new database using the oscp or osbackup database utilities described in “Maintaining the TeamConnection database” on page 177. Review the database maintenance utilities and plan your backup and recovery strategy.

## Step 2. Start the family server

You can start the TeamConnection server in the following ways:

- Double-click the testfam icon in the TeamConnection Family Administrator window.
- From the TeamConnection Family Administrator window, select the testfam icon and then select **Open → Server** from the pop-up menu. In the **testfam - Family Servers** window, select the **Start Both** push button.

After the family and notification servers are running, you can minimize the **testfam - Family Servers** window, but do not close it. Closing this window stops the servers.

- From a command prompt, change to the directory path containing the testfam database and type the following command. Use of this command assumes that the path statements and environment variables have been set properly.

```
teamcd testfam
```

See “Starting the family server” on page 86 for more information about the teamcd command.

### Step 3. Verify TeamConnection installation

The following installation verification procedure runs a command file called `univos2.cmd`, which performs a series of TeamConnection actions on testfam. Executing this command file requires that you have a full build environment set up, including VisualAge C++. “Chapter 20. Installing the build function” on page 267 explains how to install the build function. If you have not installed the build function or VisualAge C++, please omit this procedure.

To verify that TeamConnection installed correctly, do the following:

1. Type the following at a prompt in the directory where TeamConnection is installed, and then press Enter.

```
univos2
```

Type **yes** when you are asked if you want to start the build server. When you are asked to enter the build server socket name, type **bldsock**. This is the build socket name that is specified in the hosts file.

A file called `univos2.log` is created.

2. Review `univos2.log` for a completion code of zero on the commands it executes. If all commands execute successfully, then TeamConnection was installed and configured correctly.

### Step 4. Starting the TeamConnection client

To start the TeamConnection client, select the **TeamConnection Client** icon from the TeamConnection Group folder on the desktop. The Task List window appears.

---

## What do you do now?

You now have your initial family installed and running. As mentioned earlier in this chapter, you or someone else in your organization can use this family to verify that TeamConnection is working properly. You can also use this family to explore and learn about TeamConnection, and test configuration changes, user exits, automated backup utilities, and automated build and packaging events.

Before your users install the TeamConnection client, read “Chapter 8. Preparing a LAN installation for your client users” on page 81 to understand what you can do to simplify their installation procedure.

The remaining chapters of this book provide instructions for installing build server on a different machine and instructions for starting and stopping the servers.

To learn more about planning for and creating your TeamConnection production family, see “Part 3. Designing and creating your TeamConnection environment” on page 97 .



---

## Chapter 7. Installing TeamConnection for Windows

This chapter lists the hardware and software that are required before you can install and use the TeamConnection for Windows products. It also describes the tasks that you must do before installation, and provides detailed instructions for installing and configuring the TeamConnection family server and client.

### Note:

If you have installed a previous version of TeamConnection, you cannot update your old files with version 3. Instead, you need to install version 3 on a separate machine and migrate your database to version 3. See “Chapter 17. Migrating to TeamConnection version 2” on page 189 for instructions on migrating to version 3.

After you install the TeamConnection code in its directory structure (represented by x:\teamc), please keep in mind the following guidelines:

- Do not remove directories, files, or symbolic links from x:\teamc, unless you have uninstalled the product.
- Do not change the file permissions or the ownership of the directories or files in x:\teamc.
- Use x:\teamc only to store the TeamConnection code. Do not use it as the home directory for users and do not use it as the home directory for TeamConnection families.
- If this is your first installation of TeamConnection, accept the default values suggested in the installation instructions and scripts.

---

## Hardware requirements

The following are hardware requirements for the TeamConnection Windows NT **family server** and **build server** machines:

*Table 13. Hardware Requirements for a Windows TeamConnection Server*

<b>Family server</b>	<ul style="list-style-type: none"><li>• Processor: 90 MHz Pentium-based processor or higher (however, the PowerPC is not supported)</li><li>• Pointing device: A mouse or other pointing device</li><li>• Monitor: VGA or higher resolution with the appropriate adapter</li><li>• Memory: 48 MB memory minimum; more may be needed according to number of users and database size</li><li>• Disk space:<ul style="list-style-type: none"><li>– 200 MB for operating system and prerequisites</li><li>– 200 MB for user data (1+ GB recommended)</li><li>– 60 MB for TeamConnection server code</li><li>– 140 MB for ObjectStore</li><li>– 128 MB for swapper space (150+ MB recommended)</li></ul></li><li>• Communications support: Network card supported by TCP/IP for Windows NT</li><li>• A CD-ROM drive (internal or external) for installation of the product</li></ul>
----------------------	---

Table 13. Hardware Requirements for a Windows TeamConnection Server (continued)

<b>Build server</b>	<ul style="list-style-type: none"> <li>• Processor: Any processor supported by the required version of Windows, except the PowerPC</li> <li>• Pointing device: A mouse or other pointing device</li> <li>• Monitor: VGA or higher resolution with the appropriate adapter</li> <li>• Memory: 32 MB memory minimum; more may be needed according to compilers and linkers used</li> <li>• Disk space: <ul style="list-style-type: none"> <li>– 75 MB for operating system and prerequisites</li> <li>– 15 MB for TeamConnection build server code</li> <li>– 140 MB for ObjectStore</li> <li>– 45 MB for swapper space</li> </ul> </li> <li>• Communications support: Network card supported by TCP/IP for Windows NT or Windows 95</li> <li>• A CD-ROM drive (internal or external) for installation of the product</li> </ul>
---------------------	--

The following are hardware requirements for the TeamConnection Windows **client** machine:

Table 14. Hardware Requirements for a Windows TeamConnection Client

<b>Processor</b>	<ul style="list-style-type: none"> <li>• For Windows 3.1, any personal workstation with an Intel 80286 or higher processor</li> <li>• For Windows 95 and NT, any personal workstation supported by the operating system, except the PowerPC</li> </ul>
<b>Pointing device</b>	A mouse or other pointing device
<b>Monitor</b>	VGA or higher resolution with the appropriate adapter
<b>Memory</b>	12 MB minimum
<b>Disk space</b>	<ul style="list-style-type: none"> <li>• 75 MB for operating system and prerequisites</li> <li>• 25 MB for TeamConnection client code</li> <li>• 32 MB for swapper space</li> </ul>
<b>Communications support</b>	Any network card supported by the above workstation that supports TCP/IP
<b>CD-ROM drive</b>	A CD-ROM drive (internal or external) for installation of the product

## Software requirements

The following are software requirements for the Windows **family server, client, build server** machines:

Table 15. Software Requirements for TeamConnection for Windows

<b>Server</b>	<ul style="list-style-type: none"> <li>• Microsoft Windows NT, Version 3.5.1 or later, which includes TCP/IP</li> <li>• To run the installation verification tool, you need IBM Object REXX for Windows (10J9372) or the Personal REXX for Windows product available from Quercus Systems (P.O. Box 2157, Saratoga, CA 95070, Phone: 408-867-REXX)</li> </ul>
---------------	---

Table 15. Software Requirements for TeamConnection for Windows (continued)

<b>Clients</b>	<p>One of the following:</p> <ul style="list-style-type: none"> <li>• Microsoft Windows 95, which includes TCP/IP</li> <li>• Microsoft Windows NT, Version 3.5.1 or Version 4.0, which includes TCP/IP</li> <li>• Microsoft Windows, Version 3.1 or later and the following: <ul style="list-style-type: none"> <li>– DOS Version 3.3 or later</li> <li>– IBM TCP/IP for DOS Version 2.1.1 (part number 87G7184)</li> </ul> </li> </ul>
<b>Windows NT build server</b>	Microsoft Windows NT, Version 3.5.1 or later, which includes TCP/IP
<b>Windows 95 build server</b>	Microsoft Windows 95

## Preparing to install TeamConnection for Windows



If you use SmartGuides to install TeamConnection, then your TCP/IP services and hosts files are set up for you and you can skip this section.

Before you install the TeamConnection code, you must have the correct version of TCP/IP installed on your workstation. See “Software requirements” on page 70 for the communications software requirements for your operating system.

After TCP/IP is installed, update your TCP/IP services and hosts files.

Do the following steps.

1. Update the services file. If you have a services file, it is located in the system32/drivers/etc subdirectory of the Windows NT installation directory. If the services file does not exist, you must configure it through TCP/IP.

Type the following entries in your services file:

```
# TeamConnection servers
testfam    ffff/tcp    # port address for the TeamConnection test family
```

### Notes:

- a. For this installation, replace *ffff* with an appropriate port address.
  - b. Follow the line with a carriage return.
2. Update the TCP/IP hosts file. If you have a hosts file, it is located in the system32/drivers/etc subdirectory or the Windows NT installation directory. If the hosts file does not exist, you must configure it through TCP/IP.

Add the following:

- IP address.
- Server name.
- Alias name of the TeamConnection family server, which is your family name. For this initial installation of TeamConnection, the family name is *testfam*.

The following is an example of the entry you would type in your hosts file:

```
9.12.345.67    teamserv.company.com    testfam
```

**Note:** Follow the line with a carriage return. You can use the `hostname` command to get the name of the server.

3. At a prompt, type `ping testfam` to verify that you can connect to your TeamConnection family. If you receive information that is similar to the following, you can successfully connect to your TeamConnection family:

```
PINGING teamserv.company.com (9.12.345.67): with 32 bytes of data:
Reply from 9.12.345.67: bytes=32 time=10ms TTL=255
Reply from 9.12.345.67: bytes=32 time=10ms TTL=255
Reply from 9.12.345.67: bytes=32 time=10ms TTL=255
Reply from 9.12.345.67: bytes=32 time=10ms TTL=255
```

The PING command will send four requests and then it will stop.

If you receive the message `unknown host testfam`, you cannot connect to the family. Verify that the data you entered in the `hosts` and `services` files is correct, and then try the command again. If you still do not get the correct response, contact your TCP/IP administrator to solve the problem.

---

## Installing TeamConnection

This section provides step-by-step instructions for the initial installation of TeamConnection, and gives instructions for verifying the installation.

This section is intended for the administrator who does the initial install of TeamConnection for your organization. It assumes that you are installing all of the following components on the same workstation:

- TeamConnection family server (including ObjectStore database)
- Client
- Build server
- MVS build server
- CMVC migration utilities
- Online documentation

If you do not follow these instructions as written, the verification step on page “Verifying TeamConnection installation” on page 74 might fail.

The time required for installing this program varies depending on your workstation. In general, installation of the product takes about 15 minutes.

After completing the instructions in this section, you will have a family called `testfam`. Consider this your test family, and use it to verify that TeamConnection is working properly. This family is configured with the default information that IBM ships.

You can use `testfam` to explore and learn about TeamConnection. Using `testfam` with the shipped default information will help you determine how to customize TeamConnection to fit your development needs.

**Note:** When installing only a client, follow the instructions in *Getting Started with the TeamConnection Clients*.



The following is a list of the tasks you do to install TeamConnection.

Task	Page
Install the TeamConnection components	73
Verify installation	74

## Installing the TeamConnection components

This section describes the steps to follow when installing the TeamConnection components. It is assumed that you have already completed the pre-installation tasks noted in “Preparing to install TeamConnection for Windows” on page 71.

**Note:** If you have installed a previous version of TeamConnection, you cannot update your files with version 3. Instead, you need to install version 3 on a separate machine and migrate your database to version 3. See “Chapter 17. Migrating to TeamConnection version 2” on page 189 for instructions on migrating to version 3.

To install TeamConnection for Windows, follow these steps:

1. Insert the CD-ROM into the reader.
2. On your desktop, select **Start** then **Run** from the **Taskbar** or select **Run** from the File menu of the Program Manager. Type `x:\setup` in the entry field and press **OK**. Replace `x` with the CD-ROM drive letter.
3. On the Welcome window, select **Next** to advance to the next window.
4. Select **Select all** to install all the TeamConnection components. To choose components, select **Clear all** and then select the components you want installed.

**Note:** If you plan to run the installation verification procedure, be sure to install the build components.

5. Select **Next** to advance to the next window.  
The Choose Destination Location window indicates where the components you selected will be installed.
6. To change the destination directory, select **Browse**. You can type a new path in the entry field of the window displayed or select a new destination directory from the drive and directory lists. Select **OK** to return to the Choose Destination Location window, then **Next** to advance to the next window.
7. On the Select Program Folder window type the name you want to use for your TeamConnection folder or select an existing folder from the list. The default is TeamConnection. Select **Next** to advance to the next window.  
The Start Copying Files window displays. This window summarizes the components, install path, and folder you have selected.
8. Select **Next** to perform the installation.  
After the TeamConnection files have been copied to your hard drive, the TeamConnection Server Setup SmartGuide displays. This SmartGuide is a wizard that sets up your server and creates a test family.

**Note:** If you do not want to use the SmartGuide, select **Cancel** instead of **Next**. You will then be prompted to reboot.

9. Read each SmartGuide panel carefully, supply any requested information, and follow the instructions.

During setup, you may see a Startup window covering the SmartGuide window. You can move this window to the side to see the SmartGuide instructions.

Also during setup, you will be asked to restart your computer. The SmartGuide will resume its setup functions after your computer is started.

After the SmartGuide has completed, you can verify TeamConnection installation using the following procedure.

## Verifying TeamConnection installation

The following installation verification procedure runs a REXX utility called univwin, which performs a series of TeamConnection actions on testfam. Executing this command file requires that you have the following:

- A full build environment, including VisualAge C++. "Chapter 20. Installing the build function" on page 267 explains how to install the build function. If you have not installed the build function or VisualAge C++, please omit this procedure.
- The Personal REXX for Windows product.

To verify that TeamConnection installed correctly, do the following:

1. Ensure that the ObjectStore server, the family server for your test family, and the build server are running. To check that these processes are running, do the following:

### In Windows NT 4.0:

- a. Select the **ObjectStore setup** from the TeamC program list.
- b. Initialize the server.
- c. When you exit, you will be prompted to start the OS server and cache manager.
- d. Select **Yes**.

### In Windows NT 3.5:

- a. Open the Control Panel.
- b. Open the Services window.
- c. Look for ObjectStore Server R4.0 and ObjectStore Cache Manager R4.0 in the list of processes.

If these processes are not started, you can start them from the TeamConnection group folder or the Start menu of the taskbar

2. Type the following at a prompt in the directory where TeamConnection is installed, and then press Enter.

```
rexx univwin.cmd
```

A file called univwin.log is created.

3. Review univwin.log for a completion code of zero on the commands it executes. If all commands execute successfully, then TeamConnection was installed and configured correctly.

---

## Manual configuration

The following sections contain instructions for manually configuring your TeamConnection server. If you have used the TeamConnection SmartGuides to install and set up your server, these configurations have been made for you.

## Setting the environment variables

During installation, TeamConnection automatically updated your Control Panel with environment variables. If you need to change any of the environment variables, update them in your Control Panel and then restart your computer. TeamConnection places the following environment variables in the System Environment Variables section of the Control Panel.

We recommend that you use the default values for this initial installation. If you change the values, you will need to provide them when you set up the TeamConnection server.

```
HELP=x:\teamcInstallPath\nls\msg\enu
LOCPATH=x:\teamcInstallPath\locale
NLSPATH=x:\teamcInstallPath\nls\msg\enu\%N
TC_BECOME=systemLoginID
TMP=x:\teamcInstallPath\temp
IPF_PATH32=x:\teamcInstallPath
```

```
PATH=x:\teamcInstallPath\nls\msg\zzz;x:\teamcInstallPath\DLL;x:\teamcInstallPath\BIN;
TC_WWWPATH=x:\teamcInstallPath\WWW
```

Where:

- *x:\teamcInstallPath* is the directory path in which TeamConnection is installed.

For ObjectStore, the following variables are created:

```
OS_NETWORK=O4NETNP,O4NETTCP
OS_ROOTDIR=x:\lostoreInstallPath
```

Where:

- *x:\lostoreInstallPath* is the directory path in which ObjectStore is installed.

## Configuring and starting the ObjectStore server

The TeamConnection server communicates with the ObjectStore database through an ObjectStore server. This database was installed with the family server.

**Note:** You can have only one version of the ObjectStore server on the server machine.

Do the following to configure the ObjectStore server:

1. Start the ObjectStore for Windows Setup program to initialize the ObjectStore server parameters. Start the program in one of two ways:
  - From the TeamConnection Group folder on the desktop or the **Start** menu of the taskbar, select the **ObjectStore Setup Version 4.0** icon.
  - From the directory where the ObjectStore server is installed (the \bin subdirectory of the TeamConnection install path is the default), type `ossetup` and press Enter.

The ObjectStore for Windows Setup window appears. There may also appear a window asking if you want to stop the ObjectStore services since the cache manager will already be started. Select the **Stop Services** push button.

2. Select **Auto start server** if it is not already selected.

3. Select **Initialize Server**. This push button may also be named **Reinitialize Server**. The Server Initialization window appears.
4. Change the default log file's name, drive, or directory if appropriate.
5. Select **OK**. The Confirm window appears.
6. Select **Yes**. The ObjectStore for Windows Setup window appears.
7. Select **Exit**.
8. After exiting, you will be asked if you want to start the ObjectStore server and ObjectStore cache manager. Select the **Yes** push button.  
Verify that the ObjectStore server (OSSERVER) is active in the Services window in the Control Panel folder or on the Processes page of the Task Manager.

If you need to manually start the ObjectStore server, do one of the following:

- In the TeamConnection program group or folder, select the ObjectStore Server and ObjectStore Cache Manager.
- In the Services window of the Control Panel folder or on the Processes page of the Task Manager, locate the ObjectStore Server and ObjectStore Cache Manager entries in the listbox. If the status isn't started, select each entry in turn and then select **Start**.

If you experience problems configuring and starting the ObjectStore server, the following files can provide you additional information about ObjectStore errors. These files are located in the \temp directory of the TeamConnection installation path.

- osserver.txt
- osc4.out
- oss.out

## Creating a test family

Before you can verify that TeamConnection is installed correctly, you must create your family. When you create your family, you are loading default information shipped by IBM and creating a superuser ID. A superuser ID is required so that at least one person has privileged access to the family to perform special tasks, such as creating other user IDs.

This section leads you through the following procedures to create a test family called *testfam*.

1. Verifying that the database server is active
2. Creating testfam
3. Starting the family server
4. Starting the TeamConnection client

### Verifying that the database server is active

See "Verifying TeamConnection installation" on page 74 for instructions on verifying that the database server is active.

## Creating testfam

To create a test family to be used for installation verification, follow these steps. To start the family administrator program, select the TeamConnection Family Administrator from the TeamConnection group.

1. From the TeamConnection Family Administrator window, select **Family → New → Default**.

A settings notebook opens. You use this window to set up your test family. This notebook contains several pages, but for now you need only the **Family Information** and **Initial Superuser** pages. See “Part 3. Designing and creating your TeamConnection environment” on page 97 for more information on all of the family options you can set using this notebook.

**Note:** In these instructions, the values you can enter appear as follows. TeamConnection’s installation verification procedure requires that some of these values be set exactly as shown.

- **Boldface** indicates values you need to enter exactly as shown.
- *Italics* indicates values that you can substitute to suit your needs or local conventions.

2. Complete the fields on the **Family Information** page of the settings notebook as follows:

**Name** **testfam**

**Path** *d:\dbpath.*

Specify the fully-qualified path name of the directory where you want testfam stored. Make sure this directory does not already exist. TeamConnection creates testfam in a subdirectory of the path you specify. This subdirectory has the same name as the family. If you specify **c:\proddev** as the path name for testfam, for example, TeamConnection places all files related to the family in the directory path c:\proddev\testfam.

**Note:** If a testfam directory already exists, you will need to delete it before you proceed. This procedure will fail if a testfam directory already exists.

**Port** *xxxx*

Specify the TCP/IP port address for testfam, which was set in your TCP/IP services file before you installed the product.

**Mailer** **mailexit**

3. Complete the fields on the **Initial Superuser** page of the settings notebook as follows:

**Login** *userID*

Specify a user ID for the superuser for the family. For single-user platforms (OS/2), use the value set for the TC\_USER environment variable. To see this value, type the following from a command prompt and look for the TC\_USER variable:

```
set | more
```

For multiuser platforms (AIX, HP-UX, Solaris, Windows NT) specify the user’s system login ID.

**Name** *userName*

Specify the full name of the superuser, for example, Thomas C. Jones.

**Host** *hostname*

Specify the TCP/IP host name for the family server machine, which was set in your TCP/IP hosts file before you installed the product.

To see this value, type the following from a command prompt:

```
hostname
```

**Note:** You do not need to use the fully-qualified host name. You can, for example, specify *myServer* instead of *myServer.myCompany.com*.

**User** Specify the TeamConnection user ID for the superuser. If you omit this parameter, it defaults to the value specified in the **Login** field (for multi-user platforms) or to the value of the TC\_USER environment variable (for single-user platforms). It is a good idea to give the superuser an ID that is readily identifiable as a superuser. A good way to do this is to preface the user ID with su\_, such as su\_john.

**Password**

If you want to use password security, you must specify the password to be used to verify the superuser's access to the TeamConnection server. If you do not specify the password for superuser access, then no one will be able to access the database. To use password security, you need to set the **Security level** field on the **Security** page of the family settings notebook to **PASSWORD\_ONLY** or **PASSWORD\_OR\_HOST**.

4. After you have set all the values on the **Family Information** and **Initial Superuser** pages of the family settings notebook, select the **Create** push button.

As TeamConnection creates testfam, the **Create Family** window appears showing the commands executed and the status of each step in the family creation process. The following is an example of this window:  
For an explanation of the commands executed in each step of the family

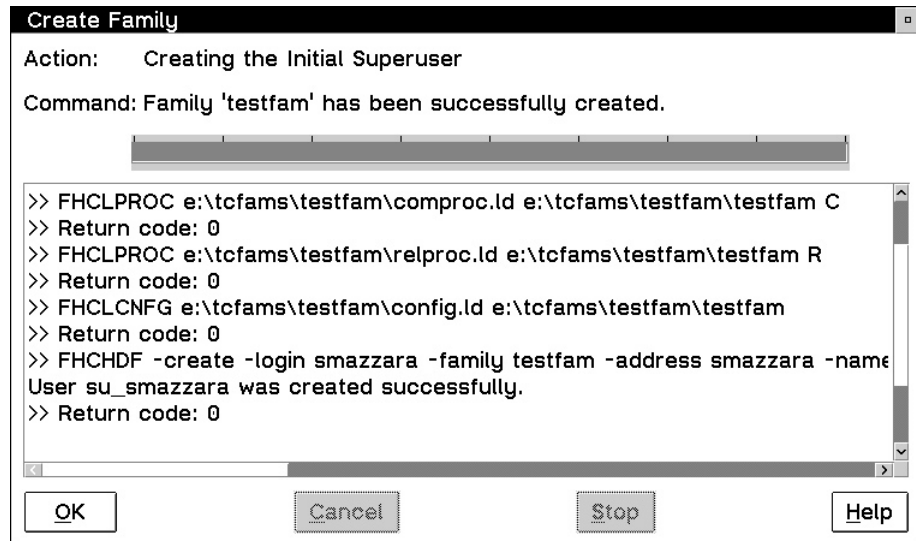


Figure 5. Create Family window

creation process, see “Appendix A. Family administration commands” on page 369 .

After testfam is created, you should be able to see the following:

- A family administrator icon labeled testfam in the **TeamConnection Family Administrator** window.
- A subdirectory called testfam in the directory path that you specified on the family settings notebook.

This subdirectory, in turn, contains the following: Copy or backup your new database using the oscp or osbackup database utilities described in “Maintaining the TeamConnection database” on page 177. Review the database maintenance utilities and plan your backup and recovery strategy.

## Starting the family server

You can start the TeamConnection server in the following ways:

- Double-click the testfam icon in the TeamConnection Family Administrator window.
- From the TeamConnection Family Administrator window, select the testfam icon and then select **Open → Server** from the pop-up menu. In the **testfam - Family Servers** window, select the **Start Both** push button.

After the family and notification servers are running, you can minimize the **testfam - Family Servers** window, but do not close it. Closing this window stops the servers.

- From a command prompt, change to the directory path containing the testfam database and type the following command. Use of this command assumes that the path statements and environment variables have been set properly.

```
teamcd testfam
```

See “Starting the family server” on page 86 for more information about the `teamcd` command.

## Starting the TeamConnection client

To start the TeamConnection client, select the **TeamConnection Client** icon from the TeamConnection Group. The TeamConnection — Tasks window appears.

---

## What do you do now?

You now have your initial family installed and running. As mentioned earlier in this chapter, you or someone else in your organization can use this family to verify that TeamConnection is working properly. You can also use this family to explore and learn about TeamConnection, and test configuration changes, user exits, automated backup utilities, and automated build and packaging events.

The remaining chapters of this book provide instructions for installing build servers and instructions for starting and stopping the servers.

To learn more about planning for and creating your TeamConnection production family, see “Part 3. Designing and creating your TeamConnection environment” on page 97 .



---

## Chapter 8. Preparing a LAN installation for your client users



This chapter applies only to OS/2 server platforms. The AIX, HP-UX, and Windows NT servers do not support LAN-based installation. From an OS/2 server, users can install the OS/2, Windows 3.1, and Windows 95 clients.

This chapter explains the following things that you can do to make the TeamConnection client installation easier for your users:

- Setting up a LAN installation
- Adding IP address to a name server

---

### Setting up a LAN installation for the client

You can set up an unattended CID install of the TeamConnection client through the LAN to enable your users to easily install the client code; they type one command rather than using the installation program's GUI.

During Step 3 of installing the TeamConnection components (see page 61), TeamConnection created a directory on your system named *x:\teamcInstallPath\client*, where *x:\teamcInstallPath* is the drive and directory in which TeamConnection is installed.

This directory contains the following files:

<b>install.exe</b>	TeamConnection installation executable file
<b>install.in_</b>	Renamed files for installation
<b>resp.dat</b>	Response file
<b>oicclnt.pkg</b>	Install package file
<b>oicclnt.icf</b>	Catalog file
<b>oicclnt.dsc</b>	Description file
<b>...</b>	Client product code

Users can access the directory from the LAN. When they type the following command from this directory, the client code is installed on their workstation.

```
install /r:x:\resp.dat /x
```

#### Where:

- **/r:x:\resp.dat** specifies the name of the response file. Substitute the current drive letter assignment for the LAN domain for *x*. The *\* is required. If it is omitted

from the response file name, the file will not be found. The TeamConnection client code is installed in the directory specified in the response file.

A response file is shipped with TeamConnection. The installation program uses the values in this file when installing the client code. You must edit resp.dat to specify your TeamConnection family name. The resp.dat file that ships with TeamConnection contains the following information:

```
COMP          = TeamConnection Client
CFGUPDATE     = auto
DELETEBACKUP  = no
FILE          = c:\teamcInstallPath
OVERWRITE     = yes
SAVEBACKUP    = no
TCFAMILY      = yourFamily
```

**Note:** In this example, *c:\teamcInstallPath* represents the drive and directory path in which TeamConnection is installed.

- **/x** specifies unattended install.

When users issue the `install` command with the `/x` option, the `config.sys` file on their machine is automatically updated. The installation program is set to install only. Therefore, it will not update, restore, or delete TeamConnection code.

## Updating the response file

The following are the parameters in the `resp.dat` file. You must update the `TCFAMILY` parameter value to the name of your TeamConnection family. You can change other parameters in this file if you want. These parameters are required.

**COMP** Do not change this value, as it must match an existing keyword in the installation program.

### CFGUPDATE

**AUTO** Automatically update the `config.sys` file

### MANUAL

Do not update the `config.sys` file

### DELETEBACKUP

**YES** Delete only backup

**NO** Delete entire product

**FILE** The new default path for the file directory

### OVERWRITE

**YES** Automatically overwrite files

**NO** Do not automatically overwrite files

### SAVEBACKUP

**YES** Save a backup version of TeamConnection when the action is update

**NO** Do not save a backup version of TeamConnection when the action is update

### TCFAMILY

The name of the TeamConnection family for your product. This value is set in your `config.sys` file, as follows:

```
set TC_FAMILY=yourFamily
```

To change other installation parameters to override TeamConnection default settings, specify one or more of the following parameters on the install command:

***/A:action***

Specifies the action you want to occur. Valid values are:

- D (delete)
- I (install)
- R (restore)
- U (update)

I (install) is the default value.

***/L1:error\_log***

Specifies the drive, path, and file name of the error log file. The drive/directory/filename path is required. The directory for the log file must already exist.

***/L2:history\_log***

Specifies the drive, path, and file name of the history log file. The drive/directory/filename path is required. The directory for the log file must already exist. If this parameter is not specified, no history log is maintained.

***/R:response\_file***

Specifies the drive, path, and file name of the specific response file.

***/S:source\_location***

Specifies the drive and path that contains the source files to install.

***/T:install\_target\_directory***

Specifies the drive and path where the product files are to be installed.

***/TU:update\_config.sys\_directory***

Specifies the drive and path of the target config.sys to be updated.

***/X***

Specifies that the action is unattended. For the install action, this is the default. If you do not specify this option, the user is prompted for any information that is needed to complete the action, progress indication is shown, and error messages are displayed in a secondary window.

---

## Adding IP address to the TCP/IP domain name server

To avoid having each user update their hosts file with the IP address, this information can be added to a TCP/IP domain name server. TeamConnection users who have access to the name server will then have the family name resolved automatically. Your TCP/IP administrator can help you do this.

See the installation chapters for more information about the IP address.



---

## Chapter 9. Starting and stopping the servers

Three TeamConnection servers are discussed in this chapter:

- The family server, which controls all the data within the TeamConnection environment
- The notification server, which sends messages between the server and the client
- The build server.

---

### Specifying the number of daemons to start

When you start the family server, you specify the number of daemons, one or more, that are to be started. A *daemon* is a process that runs as a background task and provides access to the TeamConnection database.

Because one daemon processes only one request at a time, the number of daemons you have running determines how quickly requests are processed. A daemon is not available until a request completes.

To determine the number of daemons to start, you need to understand the types of requests your users generally issue. For example, if many of the requests are for reports, which require longer processing, you will need more daemons than if most of the requests process quickly, such as checking files in and out.

Requests are queued and processed by the next available daemon. If the queue fills up, requests are not queued and the server refuses the connection. This is a signal that more daemons are needed.

If you do not explicitly specify the number of daemons when you start the family server, only one daemon is started. We recommend that you start with the cube root of the number of users. For example, start 3 daemons for 27 users, 4 daemons for 64 users. To change the number of daemons to start, you need to stop and restart the family.

Do one of the following to specify the number of daemons that you want to start:

- From the TeamConnection Group folder, display the pop-up menu for the appropriate family. Select **Open → Settings**; then following the family name in the **Parameters** field, specify the number of daemons.
- Use the Family Administrator GUI. For instructions, see “Starting servers from the Family Administrator GUI” on page 88.
- Use the teamcd command. For instructions, see “Starting the family server” on page 86 .

---

### Setting up the mail facility

TeamConnection users can receive notification when certain events occur within TeamConnection. A user's mail address is specified when a TeamConnection user ID is created. TeamConnection uses this mail address to notify users when certain actions occur.

In order for users to receive notification, the notification server must be running. When you start the notification server, you specify an executable or command file that specifies the mail exit routine that processes mail requests.

Two mail exit routine samples are shipped with TeamConnection: `mailexit.cmd` and `mailexit.exe`. These samples are located in the directory where TeamConnection is installed. Both of these samples use the `sendmail` command. You can either use one of these sample mail exits or you can use a different mail facility and write your own routine.

The `sendmail` command is part of TCP/IP, and is installed when TCP/IP is installed. If you use the `sendmail` function to send notification messages, you must configure it on your network in order for TeamConnection client workstations to receive notification messages from the server. Refer to your TCP/IP documentation for more information.

If you use a different mail facility, refer to the shipped mail exit routine sample, `mailexit.c`, to see how you can tailor TeamConnection to support your mail facility.

In order not to lose messages when the mail exit routine fails, you can have the exit routine return a code of 1041. This causes the notification daemon to exit and the mail that was being processed is not deleted. If the exit routine returns any other code, the mail that is being processed is deleted.

---

## Starting the servers

This section explains how to start the TeamConnection family server, the notification server, and the build server. These processes can be started together when you start the family server, or individually.

You can also start the servers from the TeamConnection Group Folder or the Family Administrator GUI.

## Starting the family server

You can start the family server using the `teamcd` command, an icon in the TeamConnection Group folder, or the Family Administrator GUI.

### Using `teamcd`

#### Family server and, optionally, all servers

To start the family server from the command line, type the following from a prompt:

```
teamcd [-p bldproc -a bldagnt -n mailexit -m] family n
```

Where:

- *bldproc* is the name of a file that describes the build processors that you want to start. See “Startup file for build servers” on page 94 for information about creating this file. You can also use the `TC_BUILD_PROCESSORS_FILE` environment variable to set this value.
- *bldagnt* is the name of a file that describes the build agents that you want to start. See “Startup file for build agents” on page 95 for information about creating this file. You can also use the `TC_BUILD_AGENTS_FILE` environment variable to set this value.

- *mailexit* is the executable or command file that specifies the exit routine to process mail requests. You can also use the TC\_NOTIFY\_DAEMON environment variable to set this value.
- *m* starts the family in maintenance mode. While in maintenance mode, the family is locked into read-only mode and prevents users from updating the database while maintenance is being performed.
- *family* is the name of the family you are starting.
- *n* is the number of daemons that you want to start. If this value is not typed, the default is 1. When starting build processors or the notification server, specify 0 for this parameter.

We recommend that you use this command to start your build servers. However, you can start the build server separately as described in “Starting build servers using teamcbld” on page 89.

### Build server only

TeamConnection provides build servers on the following platforms: AIX, HP-UX, OS/2, Windows NT, and Windows 95. It provides a build processor on MVS. For MVS builds, the build processor, which is installed on the MVS machine, is connected to a build server on an AIX, HP, OS/2, or Windows machine.

You can start build servers using either the *teamcd* or *teamcbld* command. We recommend you use the *teamcd* command as it provides better process and memory management of the build processors and build agents.

To start a build server apart from starting the family server type the following from a prompt:

```
teamcd -p bldproc -a bldagnt family 0
```

Where:

- *bldproc* is the name of a file that describes the build processors that you want to start. See “Startup file for build servers” on page 94 for information about creating this file. You can also use the TC\_BUILD\_PROCESSORS\_FILE environment variable to set this value.
- *bldagnt* is the name of a file that describes the build agents that you want to start. See “Startup file for build agents” on page 95 for information about creating this file. You can also use the TC\_BUILD\_AGENTS\_FILE environment variable to set this value.
- *family* is the name of the family for which you are starting the build server.
- 0 indicates that only the build server, and not the family server, is to be started. When you want to start only a build server, you must specify 0 as the number of daemons, otherwise TeamConnection will start one family daemon.

### Notification server only

You can start the notification server apart from starting the family server by typing one of the following commands from a prompt:

```
notifyd family mailexit
```

```
teamcd -n mailexit family 0
```

Where:

- *family* is the name of your family.

- *mailexit* is the executable or command file that specifies the exit routine to process mail requests. You can also use the TC\_NOTIFY\_DAEMON environment variable to set this value.
- 0 on the teamcd command indicates that only the notification server, and not the family server, is to be started. When you want to start only a notification server, you must specify 0 as the number of daemons on the teamcd command, otherwise TeamConnection will start one family daemon.

## Starting servers from the TeamConnection group folder

You can start the family or notification server from the TeamConnection Group Folder by doing the following:

### Family server

To start the family server from the TeamConnection folder, double-click on the icon that represents the family that you want to start.

Initially the TeamConnection Group folder contains an icon called **Family server** that represents the testfam family. After you create a new family, you can change the settings information for this icon and rename it to reflect your new family. Otherwise, you can use OS/2's copy function to create another icon.

### Notification server

To start the notification server from the TeamConnection folder, double-click on the **Notification Server** icon.

## Starting servers from the Family Administrator GUI

You can follow these steps to start both the family and notification servers from the Family Administrator GUI:

1. Do one of the following to display the TeamConnection Family Administrator window:
  - From the TeamConnection Group folder on the desktop, double-click on the **Family Administrator** icon.
  - Type tcadmin from a prompt.
2. Double-click the family icon for the family you want to start. The Server window appears.
3. When starting the family server, specify in the **Daemons** field the number of daemons you want started.
4. To start only one server, select the appropriate **Start** push button. To start both the family and notification servers, select the **Start Both** pushbutton.  
When a server starts successfully, the message "Press CTRL-C to stop" appears in the list box and the **Start** push button changes to **Stop**.
5. Minimize the Server window.

**Note:** Do not close the Server window. Closing the Server window stops the family server.



---

## Starting build servers using teamcbld

You can also start the build servers using the following line command:

```
teamcbld [-e environment] [-p pool] [-f family] [-c] [-s] [-l]
```

Where:

- *environment* specifies the environment that you are building for, such as OS/2 or MVS. The value you specify here can be anything you like, but it must exactly match the environment specified for a builder in order for the builder to use this build agent. This value is case-sensitive. You can also set this value using the TC\_BUILDENVIRONMENT environment variable.
- *pool* is the name of the build pool. You can also set this value using the TC\_BUILDPOOL environment variable.
- *family* is the name of the TeamConnection family. You can also set this value using the TC\_FAMILY environment variable.

If the agent is installed on a different machine than the family, this is the fully qualified name: *hostname:dbpath\family\_name.tcd*, where

- *hostname* is the name of the machine where the family resides.
- *dbpath* is the drive and path for the family.
- *family\_name* is the name of the TeamConnection family.

**Note:** You can also access the database remotely by issuing a mount command. For example, assume the build server is installed on a machine named bldsrv1, the database is installed on a machine named teamc in the root directory of the d: drive, and your family is named testfam. From bldsrv1, issue the following mount command:

```
mount x: teamc:d:\
```

Also from bldsrv1, set the TC\_DBPATH environment variable to x:\. When you issue the following command from the bldsrv1 machine, you will access the family database testfam.tcd on the d: drive of the teamc machine:

```
teamcbld -f testfam -e os2 -p pool1 -s bldsock
```

- **-c** turns caching off. For more information, see “Caching and the build directories” on page 90.
- **-s** sends log file messages to the screen. The build server generates a log file called teamcbld.log. Build server log messages can be routed to the screen using the **-s** parameter.
- **-l** increases the level of messages written to the log. This option is equivalent to the verbose option on TeamConnection teamc commands.



You can also set the **-c**, **-s**, and **-l** build options using the TC\_BUILDOPTS environment variable. See “Appendix D. Environment Variables” on page 425 for a list of TeamConnection environment variables.

## Caching and the build directories

The build server uses two special directories: the build directory and the build cache directory.

When you start a build server, a build directory is created. By default, the name of this directory is *cwd\fhbbuild*, where *cwd* is the name of the current working directory in which the build server was started. For example, if you start the build server in the directory *x:\teamcInstallPath\build* (where *x:\teamcInstallPath* is the drive and directory in which TeamConnection is installed), the build directory is *x:\teamcInstallPath\build\fhbbuild*. You can specify a different cache location by using the *-c* parameter on the *teamcbld* command. In this directory the actual compiles, links, and other data transformations invoked from build scripts are performed. The build server changes its current working directory to this directory before it invokes a build script. The name of the cache directory that you specify on the *-c* parameter must be unique.

All files needed by the build script are extracted from the TeamConnection database into the build directory or some directory subordinate to it. Because a build extracts parts from TeamConnection, anyone requesting a build needs to have PartExtract authority to all parts involved in the build. When a build event is complete, the files in the build directory are moved to the cache directory so they can be used again later. The next time a build event is processed that uses some of the same files as some previous build event, the necessary files are moved into the build directory from the cache directory rather than extracted from the TeamConnection database.

The name of this cache directory is *fhbcache*. You can specify a different name for the cache directory when you start the build server. If you are starting more than one server on the same machine, you must specify a different cache directory for each. How you do that is described in “Starting build servers using *teamcbld*” on page 89 .

A side information file called *fhbhag.\$\$\$* is maintained in the cache directory of the build server. This file contains information about the files in the cache directory. Do not delete this file, or else the cache directory will be emptied the next time the build server is started.

To control the size of the cache directory, you can set two environment variables:

```
TC_CACHESIZELIMIT=n
TC_CACHEPRUNEMETHOD=value
```

Where:

- *n* is the maximum number of bytes to allow for the build server's cache directory.
- *value* defines the order in which parts are pruned if the cache directory reaches the value of TC\_CACHESIZELIMIT. Valid values are the following:

**SIZE**    Largest parts are pruned first.

**DATE**    Least recently used parts are pruned first.

If you specify TC\_CACHESIZELIMIT but not TC\_CACHEPRUNEMETHOD, the default pruning method is by size.

## An MVS build server

To start an MVS build server, do the following to modify the RUNPGM JCL:

1. Add a job card.
2. Modify the STEPLIB DD statement to point to the data set that contains the load module teamcbld.
3. Modify the TEAMCBLD DD statement to point to the data set that will contain all your MVS build scripts.
4. Modify the EDCENV DD statement to point to the data set that contains the environment variables for the fhbmenv.var file.
5. Modify the following statement:

```
//RUNPGM EXEC PGM=TEAMCBLD,PARM='-S @nnnn -K IBM-1047 [-U unit_name] [-T]'
```

Where the:

- -S parameter indicates the port address. The address itself is preceded by an at sign (@). Ensure that the address matches the address specified in the teamagnt command for the matching build agent (or the logical address maps to the real address in your workstation TCP/IP hosts and services files).
  - -K parameter indicates the code set that text data is converted to for Because the MVS C compiler defaults to the code set IBM-1047, you need to specify the IBM-1047 code set. When starting its matching build agent, you need to specify the IBM-850 code set.
  - -U parameter indicates the default unit type for dynamic data set allocations. VIO is the default if this parameter is not specified.
  - -T parameter turns trace on.
6. Submit the job.

### The build cache data sets

As an MVS build server runs a build script, it creates a cache data set for each unique file extension type, associated with a TCEXT tag, that it encounters. When the build completes successfully, the data sets created for the build are deleted, and their contents are moved to the cache data sets.

Each cache data set is created using the attributes specified in the DD statement associated with the TCEXT attribute. (See “TeamConnection syntax for MVS build scripts” on page 293 for more information about the TCEXT attribute.) If attributes are not specified, these are the defaults used for creating a cache data set:

```
DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
UNIT=VIO
SPACE=(CYL,(30,30,1000))
```

For example, when the build server reads a DD statement that contains a TCEXT=C attribute, it allocates a partitioned data set for caching parts based on a file extension of C. If a DD statement contains the attribute TCEXT=(H,HPP), it allocates a partitioned data set for caching parts based on file extensions of H and HPP. If, in a later build event, the build server encounters another DD statement with the attribute TCEXT=C, it does not create a new cache data set. Instead, it uses the one created previously.

These cache data sets are associated with the TEAMCBLD address space. They are deleted only when the build server is stopped.

Fragmentation can occur in the cache data sets. Because TeamConnection does not provide a compression tool to manage fragmentation in these data sets, we recommend that you stop and then restart the build server when this situation occurs.

## Customizing the cache data set space attribute

The default space for cache data sets is always allocated using cylinders. The primary and secondary space allocations in the space attribute are calculated from the TC\_CACHESIZELIMIT environment variable.

To specify the size of the cache directory, you can set two environment variables:

```
TC_CACHESIZELIMIT=n  
TC_CACHEPRUNEMETHOD=value
```

Where:

- *n* is the maximum number of bytes to allow for the build server's cache directory.
- *value* defines the order in which parts are pruned if the cache directory reaches the value of TC\_CACHESIZELIMIT. Valid values are the following:

**SIZE** Largest parts are pruned first.

**DATE** Least recently used parts are pruned first.

If you specify TC\_CACHESIZELIMIT but not TC\_CACHEPRUNEMETHOD, the default pruning method is by size.

The number of cylinders allocated as primary and secondary is calculated using the following formula:

$\text{secondary} = \text{primary} = \text{TC\_CACHESIZELIMIT} / 716400$

This number is rounded up to the nearest whole number.

A minimum of 1 cylinder is allocated.

All cache data sets are allocated as partitioned data sets. The number of directory blocks allocated are based on the following formula:

$\text{directory blocks} = \text{TC\_CACHESIZELIMIT} / 22000$

This number is rounded up to the nearest whole number.

A minimum of 15 directory blocks are allocated.

---

## Starting a build agent for an MVS build server

A build agent handles access to parts data on behalf of an MVS build server. Like the family server, the build agent uses an ObjectStore database client.

There can be any number of build agents. Each is connected to one and only one build server. TeamConnection provides build agents for the following platforms: AIX, HP-UX, OS/2, and Windows NT.

There can be any number of build agents. As you start the build agents, you assign them to build pools. The environment specified in the builder for a particular build event determines which agents in the pool are available to it.

A pool is formed when you start the first build agent that specifies its name. There is no limit to the number of agents that you assign to a build pool.

If you are responsible for starting the build agents, be sure to let others who will be using them know the names of the pools and the kinds of machines assigned to them.

To start an build agent, do one of the following:

- Double-click the **TeamConnection Build Agent** icon. On the pop-up window, type the name of the startup file for the build agent you want to start. For a sample startup file, see the file `teamagnt.fil` in the `TeamConnection \bin` subdirectory. See “Creating build startup files” on page 94 for more information about creating startup files.

To start the build agent without seeing the pop-up or to change the value for the build pool, display the agent's Settings notebook. Type the appropriate values in the **Parameters** field of the Program page. Next time you double-click on the icon, the agent will start without asking for this information.

- From the command line, type the following and press Enter:

```
teamagnt -s socket_port -f familyname -p poolname -e environment
[-k local_codeset]
```

Specify the following attributes in the `teamagnt` command:

- *socket\_port* is the TCP/IP socket port. The socket port must match the socket port specified in the `teamcbld` command used to start the corresponding build server. The socket port value can have one of two formats:

- If you have added an alias name for the build socket in your hosts file, you can use that value here. For example, assume your hosts file has the following line:

```
9.12.987.65 tcserv.company.com tcfam bldsock
```

And your services file has the following line:

```
bldsock 7890/tcp # build agent
```

You can specify the socket port like this:

```
teamagnt -s bldsock -f tcfam -p pool1 -e os2
```

- If you have not specified an alias name in the hosts file, you can specify `@hostname@address`, where *hostname* is the name of the host on which the build server to which it will connect is installed, and *address* is the port address.

For example, you can specify the socket port like this:

```
teamagnt -s @bldproc1@7890 -f testfam -p pool1 -e mvs
```

- *familyname* is the name of the TeamConnection family. If the agent is installed on a different machine than the family, this is the fully qualified name: `hostname:dbpath\family_name.tcd`, where
  - *hostname* is the name of the machine where the family resides.
  - *dbpath* is the drive and path for the family.
  - *family\_name* is the name of the TeamConnection family.

**Note:** You can also access the database remotely by issuing a mount command. For example, assume the build agent is installed on a machine named

hag, the database is installed on a machine named teamc in the root directory of the d: drive, and your family is named testfam. From hag, issue the following mount command:

```
mount x: teamc:d:\
```

Also from hag, set the TC\_DBPATH environment variable to x:\. When you issue the following command from the hag machine, you will access the family database testfam.tcd on the d: drive of the teamc machine:

```
teamagt -f testfam -e os2 -p pool1 -s bldsock
```

- *poolname* is the name of the build agent pool.
- *environment* specifies the environment that you are building for, such as OS/2 or MVS. The value you specify here can be anything you like, but it must exactly match the environment specified for a builder in order for the builder to use this build agent. This value is case-sensitive.
- *local\_codeset* is the code set for data stored in TeamConnection. When starting a build agent that connects to an MVS build server, you need to specify the IBM-850 code set for this parameter. Do not use the -k flag for Windows NT build agents. Windows does not support code sets as implemented by TeamConnection and the build agent is not enabled for translation.

This command starts a process that waits for work from the TeamConnection family server, which in turn waits for work from TeamConnection clients.

---

## Creating build startup files

You can create startup files for concurrently starting build servers with the family server using the teamcd command. This is the preferred method for starting build servers. When starting the build servers in this manner, you need to create a startup file.

Information about the build servers is put in a startup file and the name of the startup file is specified in one of two ways:

- In the teamcd command using the -p or -a parameters. See page “Starting the family server” on page 86 for more information about these parameters.
- In the TC\_BUILD\_PROCESSORS\_FILE or TC\_BUILD\_AGENTS\_FILE environment variables.

You can store the build startup files wherever you like, provided that you give the full file path names for them in the -p or -a parameters, or in the TC\_BUILD\_PROCESSORS\_FILE or TC\_BUILD\_AGENTS\_FILE environment variables.

## Startup file for build servers

Describe each build server on a separate line and specify the following:

```
-s socket_port [-c cache_location -k local_codeset -n]
```

where

- *socket\_port* is the TCP/IP socket port. See “Starting build servers using teamcld” on page 89 for more about this parameter and its format.
- *cache\_location* is the fully qualified name of the directory in which caching will take place during builds. For more information, see “Caching and the build directories” on page 90.

- *local\_codeset* is the code set that text data is converted to for the build.
- *-n* specifies to erase the contents of the cache directory before starting this build server.

Lines beginning with # are considered comments. Blank lines are allowed. The following is an example of a build server startup file:

```
#-----
# My build server startup file
#-----

-s 9001 -c g:\mycache -n
```

## Startup file for build agents

Describe each build agent on a separate line and specify the following:

```
-s socket_port -e environment -p poolname [-k local_codeset]
```

where

- *socket\_port* is the TCP/IP socket port. The socket port must match the socket port specified in the build server startup file used to start the corresponding build server. See “Starting a build agent for an MVS build server” on page 92 for more about this parameter and its format.
- *poolname* is the name of the build agent pool.
- *environment* specifies the environment that you are building for, such as OS/2 or MVS. The value you specify here can be anything you like, but it must exactly match the environment specified for a builder in order for the builder to use this build agent. This value is case-sensitive.
- *local\_codeset* is the code set for data stored in TeamConnection.

Lines beginning with # are considered comments. Blank lines are allowed. The following is an example of a build agent startup file:

```
#-----
# My build agents file
#-----

-s 9002 -e OS2 -p pool1
```

---

## Stopping the servers

### The family and notification servers

You can stop the family and notification servers from the Family Administrator GUI or the command line.

- **From the GUI:**

If you started the family or notification servers from the Family Administrator GUI, follow these steps to stop them:

1. From the Family Administrator window, double click the family icon to open the Server window.
2. Select the **Stop** push button for the appropriate server to stop only one server. To stop both the family and notification servers, select the **Stop Both** push button.
3. Close the Server window.

You can also stop the TeamConnection servers using normal methods provided by your operating system, such as closing a server from the OS/2 Window List or the Windows Task List.

- **From a command line:**

1. Type the following at a prompt:

```
ps -ef | grep teamc
```

This command displays information similar to the following. The first column, labeled **PID**, is the process ID.

PID	PPID	SID	MHNDL	THRDS	MODS	SEMS	SSEG	COMMAND
232	0	32	3994	12	7	1	1	F:\TEAMC\BIN\OSCMGR4.EXE
236	19	33	3576	17	10	2	2	F:\TEAMC\BIN\OSSERVER.EXE
226	19	32	3640	3	14	2	2	F:\TEAMC\BIN\TCADMIN.EXE
241	239	32	4230	1	8	2	2	F:\TEAMC\BIN\NOTIFYD.EXE
240	238	32	3170	1	7	2	3	F:\TEAMC\BIN\TEAMCD.EXE
245	240	32	3170	1	9	2	3	F:\TEAMC\BIN\TEAMCD.EXE
244	240	32	3170	1	9	2	3	F:\TEAMC\BIN\TEAMCD.EXE
243	240	32	3170	1	9	2	3	F:\TEAMC\BIN\TEAMCD.EXE
242	240	32	3170	1	9	2	3	F:\TEAMC\BIN\TEAMCD.EXE
181	19	29	3502	1	14	2	2	F:\TEAMC\BIN\TEAMCGUI.EXE

2. Identify the process IDs that have a TeamConnection daemon running and then type the following command. The -15 option stops the server daemon gracefully.

```
kill -15 pid
```

You can issue the `kill` command with other options to recycle the daemon or stop it abnormally.

- 1 Recycles the daemon. This option stops the current request being services by the daemon and prepares it to begin receiving new requests. Use this option to reset the daemon when a request is taking too much time to complete.
- 9 Stops the daemon abnormally. Use this option only as a last resort, when no other methods work.



In UNIX environments, you can also type the following command to stop the family and notification servers:

```
stopteamc
```

## A build server

To stop a build server, do one of the following:

- Close the window in which the build server is running.
- Press Ctrl+C when the build server window has focus.
- Close the window in which the family server was started if the build server was started with the `teamcd` command.

## An MVS build server

To stop an MVS build server, cancel the RUNPGM job that was used to start it.



---

## Part 3. Designing and creating your TeamConnection environment

<b>Chapter 10. Creating your TeamConnection family</b>	99
Planning your families	99
Creating a family	99
Using the family settings notebook	100
Family information	101
Initial superuser	102
Configurable fields	103
Component processes	103
Release processes	104
User exits	104
Security	104
Authority groups	105
Interest groups	106
Creating an icon for an existing family	107
For further information	107
 <b>Chapter 11. Setting up your family structure</b>	 109
Planning your components	109
Organizing the component hierarchy	109
Determining component ownership	111
Naming the components	111
Determining access to components	112
Planning your releases	112
Relating releases with components	112
Selecting serial or concurrent development	113
Controlling database growth	113
Controlling database size automatically	114
Controlling database size manually	114
Specifying separate databases	115
Naming your releases	115
Planning your processes	115
Component processes	116
Release processes	116
How processes might change during development	118
Using the driver subprocess	119
Creating components and releases	119
Creating components	119
Creating releases	120
Creating a new release from an old release	121
 <b>Chapter 12. Preparing for your users</b>	 123
Planning for user IDs	123
Creating user IDs	124
Adding and modifying passwords	125
Planning for host lists	126
Creating host list entries	126
Planning for user access to TeamConnection data	127
What are the TeamConnection authority levels?	128
What are authority groups?	129
Creating or modifying authority groups	129
Granting authority to users	131
Granting or restricting access	132

Removing an entry from an access list . . . . .	133
Planning for user notification . . . . .	134
What are interest groups? . . . . .	134
Creating or modifying interest groups . . . . .	135
Working with notification lists . . . . .	136
Displaying interest groups . . . . .	137
Adding an entry to a notification list . . . . .	137
Removing an entry from a notification list . . . . .	138
 <b>Chapter 13. Working with configurable fields . . . . .</b>	<b>141</b>
Defining configurable field types . . . . .	142
Changing or creating configurable fields . . . . .	144
Creating and modifying configurable fields . . . . .	145
Displaying configurable field properties. . . . .	147
Changing report formats . . . . .	147
The stanza report . . . . .	147
The table report . . . . .	150
 <b>Chapter 14. Configuring family processes . . . . .</b>	<b>153</b>
Modifying or creating configurable processes . . . . .	153
 <b>Chapter 15. Providing user exits . . . . .</b>	<b>155</b>
Writing user exit programs . . . . .	155
Environment file . . . . .	161
Setting up user exits . . . . .	162
Configuring user exit parameters . . . . .	164
Sample user exit programs . . . . .	165

This section is intended for the family administrator who needs to plan for how the IBM VisualAge TeamConnection product is going to be used in the company's development environment. After the planning stage, the family administrator will use this section to learn how to do TeamConnection administrative tasks.

Before reading this section, you should be familiar with the TeamConnection terminology and concepts presented in "Part 1. Introducing IBM VisualAge TeamConnection" on page 1.

---

## Chapter 10. Creating your TeamConnection family

This chapter contains information to help you plan for your TeamConnection family. It also provides instructions for creating the family, or families, that your development organization will use.

---

### Planning your families

Careful planning of the families that your organization will use is an important first step in preparing to use TeamConnection. First, decide how many families you will need. The following will help you decide:

- Data cannot be shared between families, so group all development projects that share source data within the same family.

For example, you have several applications under maintenance or development, and these applications share some source code, such as a utility subroutine library. If you create a family for each application, each family must maintain a copy of the source code for the library. If you create one family for all the applications, they can share a single copy of the source code.

When looking at the data your projects share, consider not just the data they share today, but what they might be sharing in the future. If your development projects are going to remain separate, create individual families.

- You can create new families as your needs evolve over time.
- The more families you have, the more administrative work you will have.

Keep in mind that these are merely guidelines. If it is not clear whether you need one or more than one family, consider starting with one. You can always create another family later.

You must also decide on a name for the family. You want the name to uniquely identify the purpose of the family. For example, you might use your product name or an abbreviation of your product name. Another consideration is what case to use—lower, upper, or mixed. Mixed case is not recommended because it is more difficult to remember. You might want to ask your users what case they prefer.

---

### Creating a family

An initial family, called testfam, is configured during the installation of the TeamConnection server. This family usually serves as a test family so that you can verify that TeamConnection is working properly. You can use this family to explore and learn about TeamConnection. Eventually, you will want to create another family for use during application development.

If you create more than one family, TeamConnection places each in a separate directory. Each family requires its own audit log, user exit, mail queue, and security and configurable field information, and therefore cannot share a directory with another family.

Each family has its own unique database. If you create more than one family on a single machine, they use the same ObjectStore server to access the database. If you want to install families on separate machines, you need a TeamConnection and ObjectStore server on each machine.

At a minimum, you will provide the following information when you create your family:

- The name of the family
- The fully qualified path name of the directory where you want the family database stored. TeamConnection creates the family in a subdirectory of the path you specify. This subdirectory has the same name as the family. If you specify **c:\proddev** as the path name for the family **myfamily**, for example, TeamConnection places all files related to the family in the directory path **c:\proddev\myfamily**.
- The port address of the family server
- The login and client host information for the first superuser for the family
- The level of security to use for the family.

TeamConnection provides a Family Administrator GUI for creating families and for performing other family administrative tasks. You can access this GUI only from a family server machine.



The Family Administrator GUI is not available on UNIX platforms. See “Creating a family database” on page 369 for instructions for creating a family on UNIX platforms.

To create a TeamConnection family using the Family Administrator GUI, follow these steps:

1. Before you begin, define your family name in the TCP/IP hosts file and the port number for your database in the TCP/IP services file. See the installation instructions for information on setting up TCP/IP files.
2. Do one of the following to display the TeamConnection Family Administrator window:
  - From the TeamConnection Group folder on the desktop, double-click on the **Family Administrator** icon.
  - In Windows NT 4.0, Select **TeamConnection Family Administrator** from the **Start** menu.
  - Type **tcadmin** from a command prompt.
3. Select **New** from the Family pull-down menu; then select one of the following:
  - **Default**, to create a new database with the default values that IBM ships.
  - **Copy**, to copy an existing family’s configuration. To choose this selection, you must first select the family that you want to copy from.
4. When the family notebook appears, complete the required information about the family. After you have set values for the family in the settings notebook, select the **Create** push button.

---

## Using the family settings notebook

The following sections introduce each page of the family settings notebook. Many of these settings are discussed in greater detail in later chapters of this book.

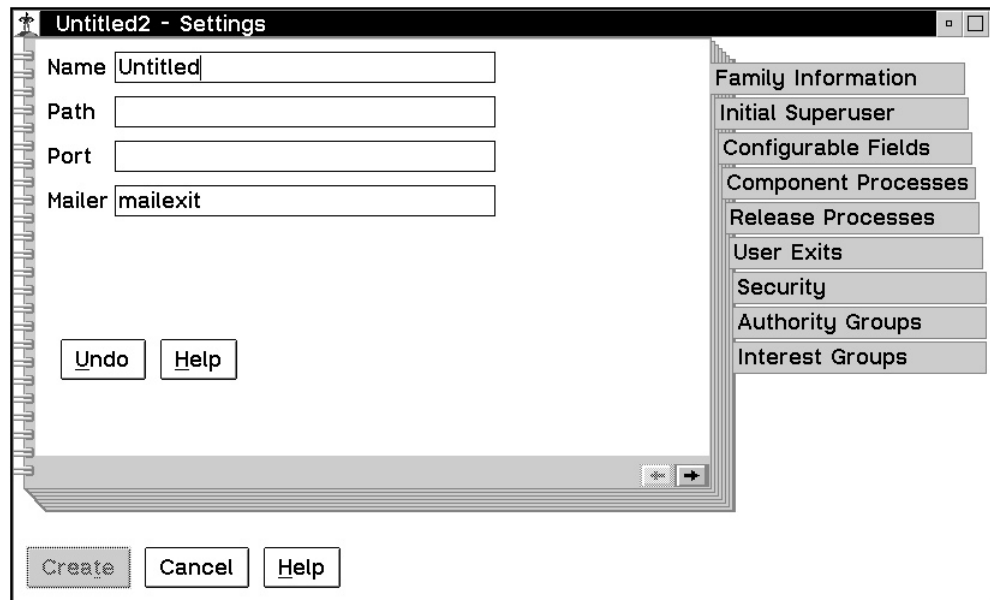


Figure 6. Family Settings notebook

## Family information

Complete the fields on the **Family Information** page of the settings notebook as follows.

**Name** Type a name for your family. The database name will consist of this name plus the extension .tcd.

**Path** Specify *d:\dbpath* for Intel platforms or **\$HOME** for UNIX platforms.

Specify the fully-qualified path name of the directory where you want the database stored. TeamConnection creates the database in a subdirectory of the path you specify. This subdirectory has the same name as the family. If you specify *c:\proddev* (Intel) or **\$HOME/proddev** (UNIX) as the path name, for example, TeamConnection places all files related to the family in the directory path *c:\proddev\yourDBName* (Intel) or **\$HOME/proddev/yourDBName** (UNIX).

**Note:** If a directory for the database name you specify already exists, you will need to delete it before you proceed. This procedure will fail if the directory already exists.

**Port** *xxxx*

Specify the TCP/IP port address that you set in your TCP/IP services file.

**Mailer** Specify the name of the mail routine you want to use to notify users of actions they need to be informed of.

TeamConnection users can receive notification when certain events occur within TeamConnection. A user's mail address is specified when a TeamConnection user ID is created. TeamConnection uses this mail address to notify users when certain actions occur.

In order for users to receive notification, the notification server must be running. The name of the mailer you specify in this field is passed to the notification server when it is started.

Two mail exit routine samples are shipped with TeamConnection: `mailexit.cmd` (OS/2 only) and `mailexit.exe`. To use the mail exit routine on Windows NT, you need to update the sample program called `mailexit.c`. These samples are located in the `\bin` subdirectory of the TeamConnection installation path. Both of these samples use the `sendmail` command. You can either use one of these sample mail exits or you can use a different mail facility and write your own routine.

The `sendmail` command is part of TCP/IP, and is installed when TCP/IP is installed. If you use the `sendmail` function to send notification messages, you must configure it on your network in order for TeamConnection client workstations to receive notification messages from the server. Refer to your TCP/IP documentation for more information.

If you use a different mail facility, refer to the shipped mail exit routine sample, `mailexit.c`, to see how you can tailor TeamConnection to support your mail facility.

In order not to lose messages when the mail exit routine fails, you can have the exit routine return a code of 1041. This causes the notification daemon to exit and the mail that was being processed is not deleted. If the exit routine returns any other code, the mail that is being processed is deleted.

## Initial superuser

Complete the fields on the **Initial Superuser** page of the settings notebook as follows.

**Login** *userID*

Specify a user ID for the superuser for the family. For Intel platforms, use the value set for the `TC_USER` environment variable. To see this value, type the following from a command prompt and look for the `TC_USER` variable:

```
set | more
```

For UNIX and Windows NT platforms, set this field to the login ID for the user.

**Name** *userName*

Specify the full name of the superuser, for example, Thomas C. Jones.

**Host** *hostname*

Specify the TCP/IP host name for the family server machine, which was set in your TCP/IP hosts file.

To see this value, type the following from a command prompt:

```
hostname
```

**Note:** You do not need to use the fully-qualified host name. You can, for example, specify *myServer* instead of *myServer.myCompany.com*.

**User** Specify the TeamConnection user ID for the superuser. If you omit this parameter, it defaults to the value specified in the **Login** field. It is a good

idea to give the superuser an ID that is readily identifiable as a superuser. A good way to do this is to preface the user ID with su\_, such as su\_john.

### Password

If you want to use password security, you must specify the password to be used to verify the superuser's access to the TeamConnection server. If you do not specify the password for superuser access, then no one will be able to access the database. To use password security, you need to set the **Security level** field on the **Security** page of the family settings notebook to **PASSWORD\_ONLY** or **PASSWORD\_OR\_HOST**.

The password must be a minimum of 1 character long and only include characters from the syntactic ASCII character set.

The password must be a minimum of 1 character long and only include characters from the syntactic ASCII character set.

If you anticipate the need for security past the basic level of authentication (HOST-ONLY) at any time in the future, it is recommended that you supply a user ID and password for the initial superuser when creating the family.

## Configurable fields

Use the **Configurable fields** page of the settings notebook to define special fields for defects, features, parts, and users. See "Chapter 13. Working with configurable fields" on page 141 for more information on using this section of the settings notebook.

## Component processes

Use the **Component processes** page to define processes and subprocesses for components defined in your family. Complete the fields on this page of the settings notebook as follows. For more information about defining and using release processes, see "Chapter 14. Configuring family processes" on page 153.

- To see the default subprocesses defined for each component process, select a process name from the **Component Processes** list. The subprocesses included will appear highlighted in the **Has these subprocesses** list.
- To add or delete subprocesses for an existing process, follow these steps:
  1. Select a process from the **Component Processes** list.
  2. To add or delete a subprocess, select it from the **Has these subprocesses** list.
  3. To save your changes, select the **Update** button.
  4. To undo your changes, select the **Undo** button.
- To create a new process, follow these steps:
  1. Position the cursor in the entry field below the **Component Processes** list.
  2. From the **Has these subprocesses** list, select the **Deselect all** push button.
  3. Clear the process name entry field and type the name of your new process.
  4. From the **Has these subprocesses** list, select the subprocesses you want to include.
  5. To add the new process to the list, select the **New** push button.
- To delete a process, select it from the **Component Processes** list and then select the **Delete** push button.

- To rename a process, follow these steps:
  1. Select a process from the **Component Processes** list.
  2. Clear its name from the entry field below the list.
  3. Type a new name in this field, and then select the **Rename** push button.

## Release processes

Use the **Release processes** page to define processes and subprocesses for releases defined in your family. Complete the fields on this page of the settings notebook as follows. For more information about defining and using release processes, see “Chapter 14. Configuring family processes” on page 153.

- To see the default subprocesses defined for each release process, select a process name from the **Release Processes** list. The subprocesses included will appear highlighted in the **Has these subprocesses** list.
- To add or delete subprocesses for an existing process, follow these steps:
  1. Select a process from the **Release Processes** list.
  2. To add or delete a subprocess, select it from the **Has these subprocesses** list.
  3. To save your changes, select the **Update** button.
  4. To undo your changes, select the **Undo** button.
- To create a new process, follow these steps:
  1. Position the cursor in the entry field below the **Release Processes** list.
  2. From the **Has these subprocesses** list, select the **Deselect all** push button.
  3. Clear the process name entry field and type the name of your new process.
  4. From the **Has these subprocesses** list, select the subprocesses you want to include.
  5. To add the new process to the list, select the **New** push button.
- To delete a process, select it from the **Release Processes** list and then select the **Delete** push button.
- To rename a process, follow these steps:
  1. Select a process from the **Release Processes** list.
  2. Clear its name from the entry field below the list.
  3. Type a new name in this field, and then select the **Rename** push button.

## User exits

Use the **User Exits** page of the settings notebook to define processes to be called at certain exit points for TeamConnection actions. See “Chapter 15. Providing user exits” on page 155 for more information on using this section of the settings notebook.

## Security

**Note:** The security features are not supported on the Windows 3.1 client.

The authentication function ensures the users of a TeamConnection family are who they claim to be. There are currently four authentication levels:

### HOST\_ONLY

A valid combination of the system login ID, TeamConnection user ID, and



host name must be used to obtain access to the family. This is the default level of security, and the level that was used in TeamConnection version 1.x.

#### **PASSWORD\_ONLY**

A user must log in to and log off of TeamConnection and supply a password in one of the following ways:

- Select **Login** from the **File** menu of the Tasks window.
- Issue the command `teamc tclogin` from a command prompt.

When the user logs in to the family, the family will send back a token associated with that user from that client. The server will check the attached token and, if valid, will proceed to perform the requested action.

This function is not currently available from the TeamConnection Windows 3.1 client.

#### **PASSWORD\_OR\_HOST**

The user can use either the **PASSWORD\_ONLY** function if he or she has a password or the **HOST\_ONLY** function if he or she has a valid host list entry. This level of security is useful for teams in which particular team members may be remote or mobile and have changing IP addresses. If the user supplies a valid password, then TeamConnection uses the password to admit access to the family. If the user either does not supply a password or supplies an incorrect password, then TeamConnection checks the user's host list entry to admit access.

The password function of this option is not currently available from the TeamConnection Windows 3.1 client.

**NONE** Any user can access TeamConnection. Neither a password nor a valid host list entry is required.

If you specify the **PASSWORD\_ONLY** option, you will need to specify a password for each TeamConnection user. See "Chapter 12. Preparing for your users" on page 123 for information on creating users.

Complete the fields on the **Security** page of the settings notebook as follows.

#### **Security level**

Select a level of security from the list box.

#### **Minimum password length**

Use this field to set the minimum number of characters to be used for passwords. The minimum and default is 1 and the maximum value is 32.

#### **Maximum invalid attempts**

Use this field to set the number of times users can attempt to log in before TeamConnection deactivates the user's ID. If this happens, a superuser must reactivate the ID before the user can attempt to log in again.

## **Authority groups**

Use the **Authority groups** page to define authority groups and actions for your family. Complete the fields on this page of the settings notebook as follows. For more information about defining and using authority groups, see "Chapter 12. Preparing for your users" on page 123.

- To see the default actions defined for each authority group, select an authority group name from the **Authority groups** list. The actions included will appear highlighted in the **Has these actions** list.
- To add or delete actions for an existing authority group, follow these steps:
  1. Select an authority group from the **Authority groups** list.
  2. To add or delete an action, select it from the **Has these actions** list.
  3. To undo your changes, select the **Undo** button.
- To create a new authority group, follow these steps:
  1. Position the cursor in the entry field below the **Authority groups** list.
  2. From the **Has these actions** list, select the **Deselect all** push button.
  3. Clear the authority group name entry field and type the name of your new authority group.
  4. From the **Has these actions** list, select the actions you want to include.
  5. To add the new authority group to the list, select the **New** push button.
- To delete an authority group, select it from the **Authority groups** list and then select the **Delete** push button.
- To rename an authority group, follow these steps:
  1. Select an authority group from the **Authority groups** list.
  2. Clear its name from the entry field below the list.
  3. Type a new name in this field, and then select the **Rename** push button.

## Interest groups

Use the **Interest groups** page to define interest groups and actions for your family. Complete the fields on this page of the settings notebook as follows. For more information about defining and using interest groups, see “Chapter 12. Preparing for your users” on page 123.

- To see the default actions defined for each interest group, select an interest group name from the **Interest groups** list. The actions included will appear highlighted in the **Has these actions** list.
- To add or delete actions for an existing interest group, follow these steps:
  1. Select an interest group from the **Interest groups** list.
  2. To add or delete an action, select it from the **Has these actions** list.
  3. To undo your changes, select the **Undo** button.
- To create a new interest group, follow these steps:
  1. Position the cursor in the entry field below the **Interest groups** list.
  2. From the **Has these actions** list, select the **Deselect all** push button.
  3. Clear the interest group name entry field and type the name of your new interest group.
  4. From the **Has these actions** list, select the actions you want to include.
  5. To add the new interest group to the list, select the **New** push button.
- To delete an interest group, select it from the **Interest groups** list and then select the **Delete** push button.
- To rename an interest group, follow these steps:
  1. Select an interest group from the **Interest groups** list.
  2. Clear its name from the entry field below the list.
  3. Type a new name in this field, and then select the **Rename** push button.

---

## Creating an icon for an existing family

You can also add an icon to the Family Administrator window for an existing TeamConnection family that was defined outside the GUI. To do this, follow these steps:

1. Select **Create icon** from the Family pull-down menu. The Locate the Database window appears.
2. Select the database file you want to use and select **OK**. An icon for that family appears.
3. After you see your family icon in the Family Administrator window, you can:
  - Start and stop the family and notification servers. See “Starting servers from the Family Administrator GUI” on page 88 for instructions.
  - Change the default values in the database to better suit your needs.

---

## For further information

The remaining chapters in Part 3. explain how to do this using the Family Administrator GUI.

For information about this task,	Go to this page.
Creating or modifying authority groups	129
Creating or modifying interest groups	135
Defining configurable field types	142
Creating configurable fields	144
Changing report formats	147
Configuring processes	153
Providing user exits	162



---

## Chapter 11. Setting up your family structure

After you create your family, it is important that you think about the following:

- How to arrange your component structure
- How to organize your releases
- What processes you want to use

This chapter helps you determine how you want to organize your family and then explains how to do it.

You need to understand what families, components, releases, and processes are and what their purpose is within TeamConnection. If you have not already done so, read “Chapter 1. An introduction to TeamConnection” on page 3, before continuing.

The following table directs you to the task you need:

For information about this task,	Go to this page.
Planning your component structure	109
Planning your releases	112
Planning your processes	115
Creating your components and releases	119

---

### Planning your components

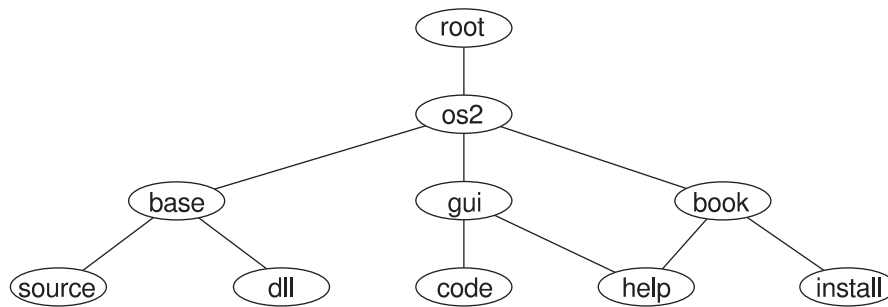
This section discusses how you can organize your component hierarchy to support your configuration management needs.

### Organizing the component hierarchy

You can organize your component hierarchy several ways. For example, one component hierarchy might mirror the application development organization hierarchy, such as department, section, team, or unit of development. Another hierarchy might reflect the software architecture of the applications under development, such as application, GUI, database.

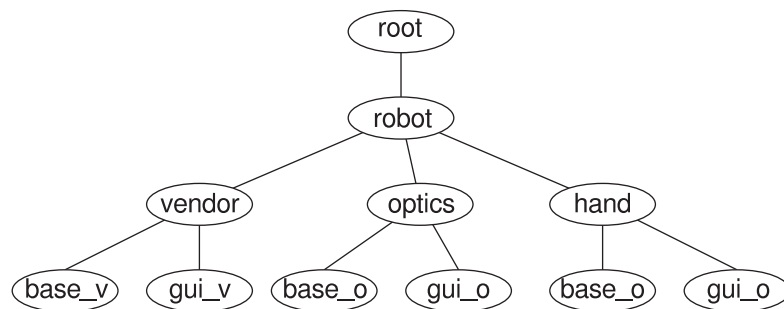
When you set up your component hierarchy, consider that all defects and features are recorded by component, and the owner of a component becomes the default owner of the defects and features for that component. This is important because defect and feature owners automatically receive a considerable amount of authority over the defects and features they own. To see the actions that defect and feature owners can perform, look at the table in “Appendix F. Authority and notification for TeamConnection actions” on page 457.

If you create your component hierarchy to store software or documentation source files, it is best to reflect the product organization at the top level. You can then create descendant components to reflect the development or maintenance responsibilities. Figure 7 on page 110 gives an example of this type of structure.



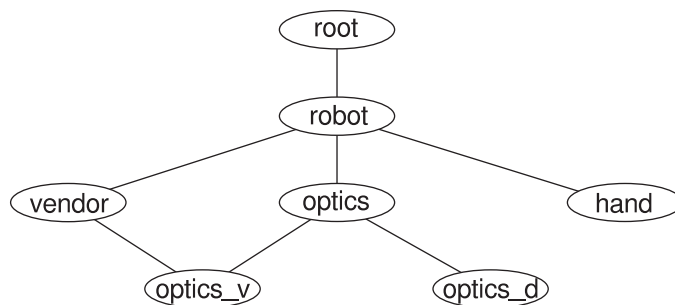
*Figure 7. A hierarchy representing product organization*

Your component hierarchy can consist of several parallel hierarchies so that you can easily restrict access to certain related components. For example, if you have vendors working on your development team, you might want to restrict their access to certain information. You can create a parallel hierarchy that contains only the information that they require. Figure 8 represents this type of structure.



*Figure 8. A hierarchy showing parallel components*

Components can have more than one parent. A component that has more than one parent inherits authority and notification from both. In Figure 9, the component optics groups both the optics\_v and optics\_d components for the development project, giving the optics\_v components two parents. The optics component manages notification for the entire optics team. Access control is managed separately for the vendors and the internal users through the lower-level components, optics\_v and optics\_d.



*Figure 9. Components with more than one parent*

Your initial component hierarchy is not necessarily going to be the same as your hierarchy a year from now. It will change as your organization grows and as your needs change. Remember that you can change the parents of a component as well as delete or rename the component.

When you plan your component hierarchy, you might find it helpful to first sketch it on paper. You can then use this sketch to help you make a table in which you note information about each component, such as the type of parts you want to control with the component, what releases a component will manage, and which processes each component and release will initially follow.

## Determining component ownership

Each component in the hierarchy has an owner. Initially that owner is the person who creates the component. After the component is created, the owner can, at any time, transfer ownership to another person.

Ownership of a component is critical. A component owner has authority to perform a wide variety of actions on that component and the parts contained in that component, as well as on all its descendant components and their parts. For example, the owner has authority to give other users access to the component and its parts and to delete the component.

You might create many of the initial components for your development organization, but you probably will not want to remain the owner of them all. As you are planning your component hierarchy, determine whom you want to own each component. The owner of the root component, the component at the top of the hierarchy, should be the person with overall responsibility for the project. If several other people have responsibility for various pieces of the development project, you might want those people to own the descendant components that relate to their piece of the project.

The owner of a parent component has the same level of authority for all of its descendant components.

Figure 10 shows a portion of a component hierarchy that Sam, a family administrator, created. Sam transferred ownership of many of the components to other members of the team. However, he kept ownership of the root component because he has overall responsibility for the project.

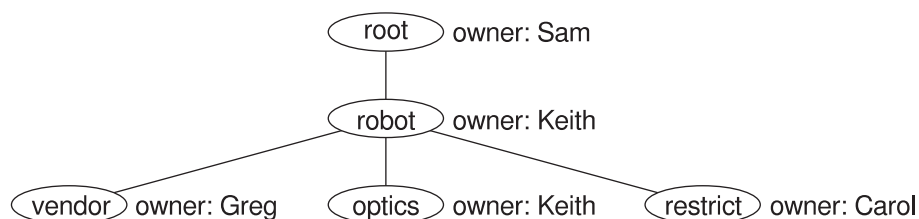


Figure 10. A hierarchy showing component ownership

## Naming the components

During the planning stage, it is helpful to decide on a component naming convention. Do you want each component name to reflect the type of data it is managing? If so, you need to understand the content, function, execution platform,

or other characteristics of the parts the component will manage. For example, the name for a component that manages data for the graphical user interface of your application might begin with the characters *gui*. Do you want component names to be in all lowercase characters, all uppercase characters, or in mixed case? The database is case sensitive. Therefore, when you are consistent with how your components are named, your users will have less difficulty finding objects in the database. The names you use must be unique within the family.

Other TeamConnection users will be able to create components, so you will want to publicize your naming convention so that everyone can adhere to it.

TeamConnection users need to access TeamConnection data, and that data can be difficult to find if they do not understand and follow your naming convention.

## Determining access to components

Each component has an access list that controls access to development data. Access authority is inherited for all descendant components, but can be explicitly restricted in the descendant components. See “Planning for user access to TeamConnection data” on page 127 for instructions. If you need to restrict access to several components for most users, you might need to redesign your component hierarchy.

---

## Planning your releases

After you decide how you are going to organize your components, determine the releases that you will initially create.

Basically, a release is a logical grouping of parts. This group of parts makes up a single version of a product, or part of a product, that is built separately, such as documentation or test cases. One release can group parts that are managed by many components.

## Relating releases with components

Every release is associated with a component that manages which users can access the parts in the release and which users are notified when certain actions occur.

For example, Figure 11 on page 113 shows the component hierarchy for a development project. Keith owns the component robot. He creates the release robot\_control to contain all the parts that pertain to the first version of the application they are developing. When Keith created the release, he specified robot as the managing component, but he did not specify an owner. Therefore, Keith is the release owner by default.



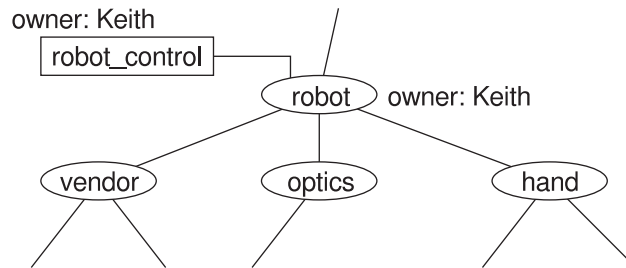


Figure 11. The release-component relationship

Keith decides that Doug should own release robot\_control. As owner of the release, Doug has authority to perform most actions against the release. However, access and notification for the release are managed by component robot and are controlled by Keith, the owner of robot. In this way, Keith can maintain access and notification control of the release, even though he has delegated the management responsibilities to Doug.

If Keith wants to give Doug the ability to control the access to release robot\_control, he can add Doug's user ID to the component's access list and specify an authority group, such as componentlead, that contains the authority to add users to access lists. Both Doug and Keith would then have the authority to add entries to the access list of component robot.

You can learn more about access lists and authority groups in "Planning for user access to TeamConnection data" on page 127.

## Selecting serial or concurrent development

The release can be set up for developing in serial development or in concurrent development mode. In *serial development*, a part is locked when a user checks it out, and no one else can update the part as long as it is checked out. In *concurrent development*, more than one user can simultaneously have the same part checked out. When TeamConnection detects that someone else has made changes to a part that another is checking in, it notifies the user that a collision has occurred. The user can reconcile the changes using the TeamConnection merge program.

You specify the mode in which your users will work when you create the release. Be aware, however, that after the mode is set to concurrent, you cannot change it back to serial. However, it is possible to change the mode from serial to concurrent.

## Controlling database growth

TeamConnection places a 2-gigabyte size restriction on its databases. To keep from reaching this limit, consider placing larger releases on separate databases. You can control the size of your database in the following ways:

- Automatically by setting options when you create a release
- Manually by pruning a release

The following sections explain these methods of controlling database growth. If, after taking these measures, you reach the 2-gigabyte size limitation, you can create a raw file system (rawfs) database. This allows you to distribute the

TeamConnection database across several disks. See “Using a raw file system (rawfs) database” on page 187 for more information.

## Controlling database size automatically

You can help control the size of your family database by requesting the following options when creating a release:

- Automatic pruning of work areas
- Maximum number of build output versions

These options let you reclaim database space without extra work.

**Automatic pruning of workareas:** To understand automatic pruning of work areas, it helps to understand the basics of work area versioning. Every time you freeze a work area, TeamConnection saves a revision level of the work area. When you freeze work area 123, for example, a version called 123:2 is created. This version contains information about each part in the work area and its current version at the time the work area was frozen. It may contain version 1 of part optics.c, for example. If you freeze the work area again later, a new version called 123:3 is created with information about the versions of the parts in the work area when it was frozen. This version may contain version 2 of part optics.c. Each of these work area versions is saved in the database and you can retrieve the versions of the parts they contain before you integrate the work area into the release.

Automatic pruning enables you to delete all versions of work areas after you have integrated the most current version of the work area into a release. You can indicate whether you want automatic pruning of work areas by doing one of the following:

- Select **Automatic version pruning** on the Create Releases window when you create the release
- Use the + autopruning flag with the release command

These options tell TeamConnection to destroy work area versions when a user integrates a work area or commits a driver to the release.

Be aware that when work areas are destroyed, most of their change tracking information is also destroyed and it will be more difficult for you to go back to previous versions.

**Maximum committed output versions:** You can also indicate the maximum number of committed build output versions you want kept. When that number is reached, TeamConnection discards the oldest one. Otherwise, all build outputs are saved. You can set the maximum committed build output versions by doing one of the following:

- Specify the number you want kept in the **Maximum number of output versions** field on the Create Releases window
- Use the -outputVersions flag in the release command

## Controlling database size manually

If you choose not to use autopruning to control database size, you can still prune your releases manually as follows:

1. From a Releases window, select the release you want to prune and then select **Prune** from the **Selected** menu.

2. In the **Version branch** field of the Prune Release window, type the name of the first version of the branch associated with the work area you want to prune and then select **OK**.

## Specifying separate databases

One way of controlling database size is to use release databases. Objects shared by all releases (such as users, defects, features, and host lists) are stored in the family database. All other objects can be stored in separate databases, one for each release. It is much easier to manage the space utilized by releases when they are isolated to their own release database. Specifying separate databases for heavily used releases can help keep a database within the 2-gigabyte size limit. Separate databases can reside on the same workstation or be spread across remote machines for better performance.

You can specify a separate database for the part data when the release is created. Use the `-db` flag in the release command or the **Database** field on the Create Releases window to specify the database you want the release to use. The directory path you specify is the name of the directory on the server. You can give the database any name, either specifying an extension or not. If you do not specify a directory path, TeamConnection puts the database in the directory pointed to by the `TC_DBPATH` environment variable.

## Naming your releases

Next, decide how you are going to name the releases you create. You might want to name your releases according to the product or object you are building. For example, *prod1r1* for release 1.1 of your application, or *using1r1* for the book files for release 1.1 of your application. To make it easier on your users, continue using the basic naming convention that you are using for your components. The names you use must be unique within the family.

---

## Planning your processes

Before you create your family's components and releases, decide what processes you are going to initially use during development.

A TeamConnection process is used to enforce a specific level of control of components and releases. TeamConnection is shipped with a set of predefined processes for both components and releases, so you provide different processes for each to follow. You can use these processes, or you can configure your own processes using some of the predefined subprocesses. "Chapter 14. Configuring family processes" on page 153 explains how you configure TeamConnection processes.

You probably already have a process for tracking problems as well as a process for tracking suggested improvements to your applications. If you want to continue to use those processes, determine how you can best group the TeamConnection subprocesses to reflect your current process. If you do not have an existing method, decide how tightly you want to control part changes and track defects and features.

The poster, *Staying on Track with TeamConnection Processes*, explains the various states that different TeamConnection objects can go through depending on the

process that is being followed. You might want to study this information before you determine how you want to use TeamConnection processes.

## Component processes

A component's process determines how much planning and designing is required before work on a defect or feature begins and whether the originator is required to verify that the work was done correctly.

When choosing a process for a component to follow, think about the type of data within the component. For example, the parts within one component might contain complex code that is time-consuming to fix. Before any defects or features are accepted, the work needs to be designed and sized, so the *preship* process is followed. Parts within another component contain code that is relatively easy to fix and test. The defects and features for this component do not need to be designed and sized, so the *prototype* process, which contains no subprocesses, is followed.

For components, you can require users to follow any, all, or none of the following predefined subprocesses:

### **dsrDefect**

Design, size and review fixes to be made for defects

### **verifyDefect**

Verify that the fixes work

### **dsrFeature**

Design, size, and review changes to be made for features

### **verifyFeature**

Verify that the features have been implemented correctly

The following table lists the component processes that are supplied by IBM. Each process combines a set of TeamConnection subprocesses. An X under the TeamConnection subprocess indicates that the corresponding process includes it.

Table 16. Shipped component processes

Shipped TeamConnection component process	TeamConnection subprocesses			
	dsrDefect	dsrFeature	verifyDefect	verifyFeature
default		x	x	x
development		x		
emergency_fix				
maintenance			x	x
preship	x	x	x	x
prototype				
test		x	x	x

## Release processes

A release's process determines to what extent part changes are tracked and the procedure for integrating changed parts into a build. Release processes control the day-to-day work that is involved in producing the product-fixing defects and

implementing features, as well as building the product. The type of process control you want to enforce on a release is likely to change over time.

For releases, you can require any, all, or none of the following predefined subprocesses:

**track** This subprocess is TeamConnection's way of relating all part changes to a specific defect or feature and a specific release. Each work area gathers all the parts modified for the specified defect or feature in one release and records the status of the defect or feature. The work area moves through successive states during its life cycle. The TeamConnection actions that you can perform against a work area depend on its current state.

You must use the track subprocess if you want to use any of the other release subprocesses.

#### **approval**

This subprocess ensures that a designated approver agrees with the decision to incorporate changes into a particular release and electronically signs a record. As soon as approval is given, the changes can be made.

**fix** This subprocess ensures that as users check in parts associated with a work area, an action is taken to indicate that they have completed their portion. When everyone finishes, the owner of the fix record (usually the component owner) can change the fix record to complete. The parts are then ready for integration.

**driver** A driver is a collection of all the work areas that are to be integrated with each other and with the unchanged parts in the release at a particular time. The driver subprocess allows you to include these changes incrementally so that their impact can be evaluated and verified before additional changes are incorporated. Each work area that is included in a driver is called a driver member.

**test** The test subprocess guarantees that testing occurs prior to verifying that the fix is correct within the release.

The following table lists the release processes that are supplied by IBM. Each process combines different sets of TeamConnection subprocesses. An X under the TeamConnection subprocess indicates that the corresponding process includes it.

Table 17. Shipped release processes

Shipped TeamConnection release processes	TeamConnection subprocesses				
	track	approval	fix	driver	test
prototype					
development	x		x		x
test	x		x	x	x
preship	x	x	x	x	x
maintenance	x		x	x	x
emergency_fix	x			x	
track_only	x				
track_driver	x		x	x	
track_approval	x	x	x	x	

Table 17. Shipped release processes (continued)

Shipped TeamConnection release processes	TeamConnection subprocesses				
	track	approval	fix	driver	test
track_test	x		x	x	x
track_full	x	x	x	x	x
no_track					

It is important that your users understand the meaning of each process and the type of control it enforces. For example, if a stringent release process such as `track_full` is selected, actions have to occur in a precise order. Compare this to the `no_track` process where users can freely check parts in and out of TeamConnection.

## How processes might change during development

TeamConnection provides different processes for components and releases. The processes you choose depend on how tightly you want to control changes and how you want to handle defects and features. Your choices, of course, will vary depending on where you are in your current development cycle. You can change your processes during a development effort to reflect different phases. For example, you might do the following:

- During the requirements gathering phase, you create a component that manages the requirements documentation. You want minimal defect or feature processing against the parts managed by this component, so you select a process, such as *prototype*, that is not strict. The release would also follow a relaxed process, such as *prototype*.
- After the requirements are settled and design work begins, you want to control changes to the requirements data but not to the rapidly evolving design data. For the requirements component, you change to a process that includes review and verification, such as the *default* process. You also create a new component to manage the design documentation and you select a process that is not strict. You continue to use the *prototype* process for the release.
- When coding begins, you change the process for the design component to one that includes review and verification, such as *development*. You also create a new component to manage the code files. Because you will be loading files into TeamConnection rapidly, you select a process that is not strict, such as *prototype*.

You also change to a release process, such as *development*, that tracks the resolution of defects and features.

- After all the code files that are managed by a given component pass unit test, you change that component's process to one that includes review and verification, such as *default*. You also change the release process to one with tight control, such as *track\_full*, so that you can carefully manage code changes.
- Ninety days before your delivery date, you change all the components to a very stringent process, such as *preship*, to ensure that all new features or defects are reviewed for impact to the delivery schedule.

## Using the driver subprocess

The driver subprocess is a way to better control the building and testing of your application code. As you develop your application program, you will probably have many drivers within a release, and you can have multiple overlapping releases during a development cycle.

For example, let's say you are developing a robot application and you send monthly updates to customers for their feedback. You do regular driver builds of your application. You use the driver subprocess to help you control the integration of changes that occur between builds. At some point during the month you cut off changes to the current release r9504 for system testing. You are then ready to create a new release called r9505. Using TeamConnection, you can link the parts in r9504 to the new release r9505. During the follow-on development work, release r9504 is still there. At the end of the month you send your final build and tested driver of release r9504 to your customers.

The following figure depicts this process graphically. In this illustration, releases are labeled *ryymm*, where *yy* represents the year (such as 95 for 1995) and *mm* represents the month (such as 04 for April). Drivers are labeled *dmdd*, where *m* represents the month (such as 4 for April) and *dd* represents the day of the month.

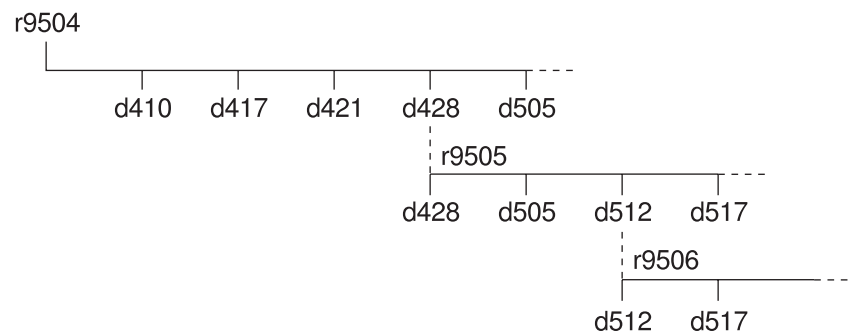


Figure 12. Using the driver subprocess

---

## Creating components and releases

Now that you have gone through the planning phase and have your structure on paper, you are ready to create the components and releases that your organization will use.

### Creating components

For each family you create, TeamConnection creates the top component called root. Therefore, at least the first component you create has root as the parent. Do not change the name of the root component.

As with most TeamConnection tasks, you can use either the GUI or the command line. To create a component, do one of the following from a client machine:

- From the GUI:
  1. From the Tasks window, select **Components → Create** from the Actions pull-down menu. The Create Components window appears.

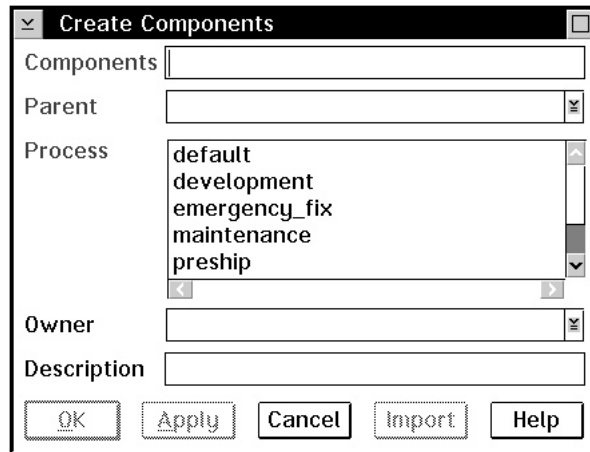


Figure 13. Create Components Window

2. Type the component name, the parent name, and the name of the process you want the component to follow. Other information on this window is optional.
  3. Select **OK** to create the component and close the window, or select **Apply** to create the component and leave the window open.
- From a command line, type:
 

```
teamc component -create componentName -parent parentName
               -process processName
```

For more information about the component command, refer to the *Commands Reference*

## Creating releases

To create a release, do one of the following from a client machine. Before creating a release, read “Planning your releases” on page 112.

- From the GUI:
  1. From the Tasks window, select **Releases** → **Create** from the Actions pull-down menu. The Create Releases window appears.



Figure 14. Create Releases Window

2. Type the release name and the name of the managing component, and select the process you want the release to follow.

If you want your users to work on parts concurrently, check **Concurrent development mode**. For serial development, leave this box unchecked. Be aware that when you select a development mode, you cannot change it.

To reclaim database space without extra work, specify that you want automatic pruning of work area versions that have not been integrated with the release. Another way to save database space is to specify the maximum number of build output versions you want kept.

You can also specify a database that is separate from the family database.

Refer to “Planning your releases” on page 112 for more information about these options.

3. Select **OK** to create the release and close the window, or select **Apply** to create the release and leave the window open.

- From a command line, type:

```
teamc release -create releaseName -component componentName
               -process processName [-concurrent] [+autopruning]
               [-outputVersions number]
```

For more information about the release command, refer to the *Commands Reference*.

## Creating a new release from an old release

A TeamConnection family contains the work of many individuals for one product or project. Within that family, several releases can be created.

For example, currently, a team is working in the release `robot_control`. After this team finishes with the current edition of the robot project, the next team might work on a follow-on release of the robot product. That team could create a new release called `robot_v2` in which to work. Another possibility is a team that wants to implement the `robot_control` program on a different type of robot (similar to developing the same application for a different operating system). The team could create a release called `robot_mk5` in which to work. These various releases in a family are used to isolate changes to a similar code base.

These examples illustrate that the various releases in a family will often share a code base from another release. Administrators can link releases in order to share code between the linked releases.

For example, to create a new release called `robot_v2` that links to release `robot_control`, do the following using either the GUI or command line interface:

1. Create the new release `robot_v2`.
2. Create a work area.
3. Link the new release to `robot_control` using the work area that was just created.
4. Integrate the work area with the new release.

---

## Chapter 12. Preparing for your users

This chapter helps you determine how to identify your users to TeamConnection, how to set up your host lists, how to set user passwords, and how to set up authority groups and interest groups.

---

### Planning for user IDs

Before you start defining users, make sure you have read “Chapter 10. Creating your TeamConnection family” on page 99 and understand how TeamConnection implements security for families.

Each user must have a TeamConnection user ID that uniquely identifies the user to TeamConnection and gives the user access to TeamConnection objects. TeamConnection uses the following terms and objects to identify users and control their access to TeamConnection information:

#### User ID

The ID by which TeamConnection knows you and assigns access authority to you, and the ID under which you issue TeamConnection commands. This ID is the one specified by the TC\_BECOME environment variable or on the **Become user** field of the TeamConnection settings notebook.

#### Login ID

This term is most meaningful in a multiuser environment, such as AIX, HP-UX, or Windows NT. The login ID is the ID that you use to log in to your workstation and is specified on the **User ID** field of the TeamConnection settings notebook. In single-user environments, such as OS/2 and Windows 3.1 the login ID is the one specified by the TC\_USER environment variable. In Windows 95, the user ID is used if one is specified, otherwise, the TC\_USER environment variable is used.

#### Host name

The name that identifies the workstation you are logged in to. This is the TCP/IP host name of the machine from which you access TeamConnection.

#### Host list

A TeamConnection object that associates user IDs, login IDs, and host names. Each TeamConnection user has a host list that controls which user IDs, host names, and login IDs he or she can use to access TeamConnection.

The following example illustrates how these terms and objects are put into use and why TeamConnection requires both a login ID and a user ID:

A user named Chris Wright has the following TeamConnection responsibilities:

- Develops code for a product
- Performs superuser tasks for the family in which the product is developed
- Extracts releases to deliver the code to translators

Chris has a workstation with the TCP/IP host name *cwright.company.com*. This workstation is used for Chris's daily programming activities and for superuser activities. For day-to-day programming work, Chris uses the TeamConnection user ID *cwright* from host name *cwright.company.com*. For superuser activities, Chris uses the TeamConnection user ID *su\_cwright* from host name *cwright.company.com*. For translation activities, Chris has access to a workstation

with the TCP/IP host name *nlsserver.company.com*. Chris logs into *nlsserver.company.com* as *cwright* and extracts files from TeamConnection as user *cwright*. To enable Chris to perform each of these activities with the proper TeamConnection authority, Chris needs the following TeamConnection host list entries:

Table 18. Host list entries for a sample user

Login ID	Host name	User ID
cwright	cwright.company.com	cwright
cwright	cwright.company.com	su_cwright
cwright	nlsserver.company.com	cwright

With these host list entries, Chris can do the following:

- Access TeamConnection as user ID *cwright* from to perform daily programming tasks from workstation *cwright.company.com*.
- Access TeamConnection as user ID *su\_cwright* to perform superuser tasks from workstation *cwright.company.com*.
- Access TeamConnection as user ID *cwright* to perform NLS activities from workstation *nlsserver.company.com*.

---

## Creating user IDs

You can use a number of methods to assign IDs to your users. For instance, you can have each user's TeamConnection ID match the user's login ID. This is easy to do because each user already has a login ID. This method can be ideal if your users use the same login ID across their different systems, but it's confusing if they do not. For example, if Chris Wright has access to two workstations and logs in to one as *chris* and the other as *wright*, then identifying all the objects that belong to Chris Wright is more complicated. If you use this method, when a user moves on to another project, you will have to transfer the ownership of objects from that user to another user. Using this method can make additional work for you on a project where people move around a lot.

Another method is to assign IDs according to the roles that people have, such as *proj\_lead*, *writer1*, *tester\_mvs*, and *manager*. When you use this method, ownership remains the same when people leave the project. However, it can be more difficult to identify the person who owns a particular ID. For example, it is easier to identify Chris Wright as the user of the ID *cwright* than it is to identify him as the owner of the ID *writer1*.

You must have superuser authority to create user IDs. To create a new user ID in TeamConnection, do one of the following from a client machine:

- From the user interface:
  1. Select **Users → Create** from the Actions pull-down menu on the Tasks window. The Create User window appears.

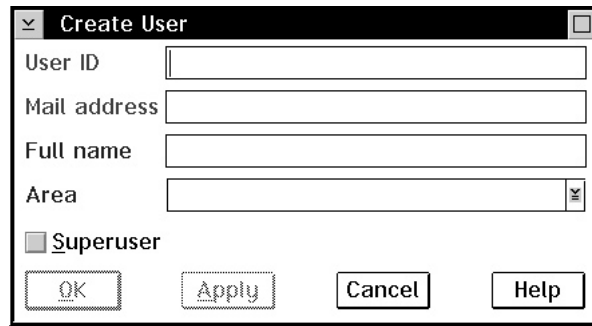


Figure 15. Create User window

2. Type the user's ID and electronic mailing address. Other information on this window is optional.  
For more information, select **Help** from the Create User window.
  3. Select **OK** to exit the window, or select **Apply** when you want to add another user ID immediately.
- From a command line, type:  

```
teamc user -create -login userID -name name -address mailAddress
```

For more information about the user command, refer to the *Commands Reference* book.

Superuser privilege is granted to one user ID when TeamConnection is installed. This privilege is required so that at least one person has privileged access to the family to perform special tasks, such as creating and deleting other user IDs. The person with superuser privilege can perform all possible actions in your TeamConnection family. This is an authority level that you definitely want to limit to only a very few individuals. In fact, individuals who have a superuser ID should also have another ID that has less authority, which is the ID they will use when doing their normal work. The user can switch between the two IDs by using the TC\_BECOME environment variable.

To give a user superuser privilege, include the +super flag with the user command, or select the **Superuser** check box when using the GUI. When creating superuser IDs, you might want to begin the ID with su\_. This will make it obvious as to which ID has superuser privilege. You must have superuser privilege yourself in order to give this authority to someone else.

**Note:** Before TeamConnection recognizes a new user, you must add at least one host address entry to the user's host list. "Planning for host lists" on page 126 provides more information.

---

## Adding and modifying passwords

If you are using PASSWORD\_ONLY or PASSWORD\_OR\_HOST level security, you will need to modify the user IDs you have created to add passwords to them. To add or modify a user ID's password, follow these steps:

- From the user interface:

1. Select **Users → Modify → Password** from the Actions pull-down menu on the Tasks window. The Modify Password window appears.
2. Type the user ID in the **User ID field**.
3. Type the password in the **Password** and **Verify Password** fields.

**Note:** The default minimum password length is 8 character. To determine if the the minimum password length for a family is something other than the default, check the **Security** page of the settings notebook for the family.

4. If you are modifying an existing password, type the old password in the **Old Password** field.
  5. To apply the changes and leave the Modify Password window open, select the **Apply** push button. To apply the changes and close the window, select the **OK** push button.
- From a command line, type:

```
teamc user -modify -login userID -password password
```

For more information about the user command, refer to the *Commands Reference*.

---

## Planning for host lists

When host-based security is in effect, each user ID is associated with a host list, which is a list of client machine addresses from which the user can access TeamConnection when using that ID. Users must have at least one entry on their host lists so that TeamConnection will recognize them as valid users.

A superuser must create the initial host entry for a user. After the initial entry is created, users can add host entries for themselves. Additional entries in a host list lets a user access TeamConnection from other client machines.

---

## Creating host list entries

The initial host list entry for each user must be created by someone with superuser authority. To create a host list entry in TeamConnection, do one of the following from a client machine:

- From the user interface:
  1. Select **Users → Add host** from the Actions pull-down menu on the Tasks window. The Add Host window appears.

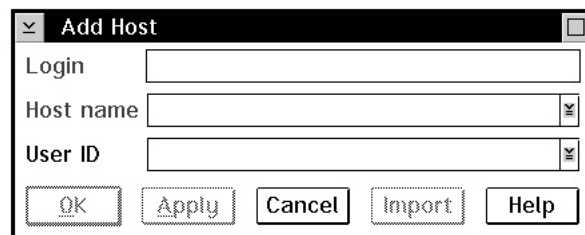


Figure 16. Add Host window

2. Type the login ID and the name of the client machine from which the user will access TeamConnection. The user ID is optional.

For more information, select **Help** from the Add Host window.

3. Select **OK** to exit the window or select **Apply** when you want to add another host list entry immediately.

- From a command line, type:

```
teamc host -create login@hostName -login userID
```

The login value following -create is the user ID (TC\_USER) with the host name appended to it, while the -login attribute flag is the TeamConnection user's ID (TC\_BECOME) for the GUI (usually, these values are the same). The value of the user's ID is case sensitive, so type it exactly as it was typed when the user was created.

For more information about the host command, refer to the *Commands Reference*.

---

## Planning for user access to TeamConnection data

As soon as a TeamConnection user ID and a host list entry ( in a host-based authentication scheme) are created for a user, that user automatically has the authority to perform certain basic actions within the family. This authority is referred to as *base authority*. Beyond base authority, authority to access TeamConnection data is managed by the components that you create. Each component has an *access list* that controls access to development data. Authority granted in an access list is called *explicit authority*. Explicit authority is inherited by descendant components. So when a user has authority to perform actions within one component, that authority is inherited for all its descendant components. Explicit authority and how it is inherited is discussed in "Granting authority to users" on page 131 .

TeamConnection users also get authority to perform additional actions when they own TeamConnection objects. Authority granted by ownership is called *implicit authority*. Because this authority is inherited, you need to be careful when assigning component ownership and when granting access authority to your users.

Figure 17 on page 128 shows Doug as the owner of the optics component. As owner, Doug has the implicit authority to perform most TeamConnection actions against the objects that are managed by the optics component. Doug wants Greg to be able to perform many of the same actions, so he gives him explicit releaselead authority through the component's access list. Because authority is inherited by descendant components, Doug and Greg have releaselead authority in the optics\_v and base\_h components.

Access list			
component	user ID	authority	type
team_a	doug	releaselead	granted
team_a	greg	releaselead	granted

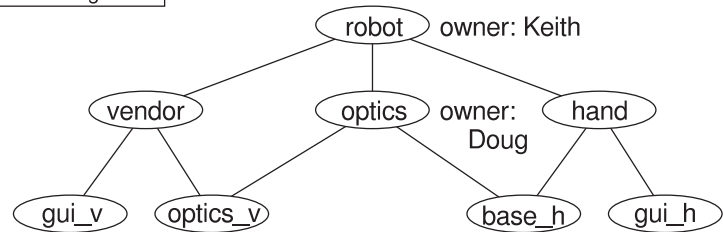


Figure 17. Granting authority to other users

Look at your component hierarchy before granting access authority. Do you want that user to have the authority to perform the same set of actions in all of the descendant components? If the answer is no for only one or two of the components, you can restrict the user from inheriting authority for those components. See “Granting or restricting access” on page 132 for instructions on restricting access to a component. If the answer is no for many of the descendant components, you might not want to give the user that level of authority.

## What are the TeamConnection authority levels?

The authority to perform various TeamConnection actions is based on four types of authority levels: base authority, implicit authority, explicit authority, and superuser privilege. These types of authority are described as follows. For a summary of the authority required for performing TeamConnection actions, see “Appendix F. Authority and notification for TeamConnection actions” on page 457.

### Base authority

All users defined to TeamConnection have authority to perform the following actions:

- Open defects and features
- Modify the information for their user ID
- Display information about any user ID
- Add notes to existing defects and features
- Search for information within TeamConnection to create reports

### Implicit authority

Many TeamConnection objects, such as a component, part, or defect, have an owner. The object owner automatically receives authority to perform certain actions. For example, when a defect is opened, the owner has the authority to accept the defect or reassign ownership of the defect. Similarly, the owner of a component, a release, or a feature has authority related specifically to those objects. Sometimes authority is given based on an action the user takes. For example, when someone checks a part out of TeamConnection, that person is given the authority to check it back in.

### Explicit authority

Some users need additional authority to perform actions against objects that they do not own. For example, users other than the component owner



will need to check parts out of TeamConnection. The component owners give additional authority to users by adding their names to the component's access list.

See “Granting authority to users” on page 131 for more information about access lists.

### **Superuser privilege**

A user with TeamConnection superuser privilege can perform any TeamConnection action. Only an individual with superuser privilege can add, delete, or re-create a user ID, as well as grant superuser privilege to another user. Only a few users in your organization should have this privilege.

See page 125 for information about granting this privilege.

## **What are authority groups?**

There are many actions that users can perform against TeamConnection objects. It would be tedious to grant one action at a time to each of your users. Instead, you can grant a user the authority to perform a group of actions, called an *authority group*. For example, the managers of a project might want to view only the status of certain TeamConnection objects, while the developers need to view objects and also check in, check out, and extract parts. You can grant access to an authority group for either of these jobs.

As family administrator, you are responsible for the authority groups that your organization uses. IBM ships a set of default groups with TeamConnection. You can use these groups as is, modify them, or create new groups.

---

## **Creating or modifying authority groups**

When the database is initially created, the authority table contains the default values for the authority groups. If the default authority groups are not adequate for your development organization, you can create new authority groups or modify existing groups. Authority groups can be created or modified at any time during your development cycle.

First, decide what group of actions the intended users are required to perform. If there is a shipped authority group that closely matches your needs, you might want to modify that group. Otherwise, you will need to create a new group. To help you keep track of the authority groups that the family uses, add the name of each authority group you create to the worksheet provided in “Appendix H. Worksheets” on page 479.

We recommend you use the Family Administrator GUI to create or modify authority groups; however, if you prefer to do this manually, see page 371. Instructions for using the GUI follow.



Go to “Creating or modifying authority groups” on page 371 to complete this task.

Before you do this task, we recommend that you stop the family server (see page “Stopping the servers” on page 95 for instructions).

1. Do one of the following from a server machine to display the TeamConnection Family Administrator window:
  - From the TeamConnection Group folder on the desktop, double-click on the **Family Administrator** icon.
  - Type `tcadmin` from an OS/2 command prompt.
2. Display the family icon's pop-up menu; then select **Settings**. The Settings notebook appears.
3. Select the **Authority Groups** page to display the authority group settings.

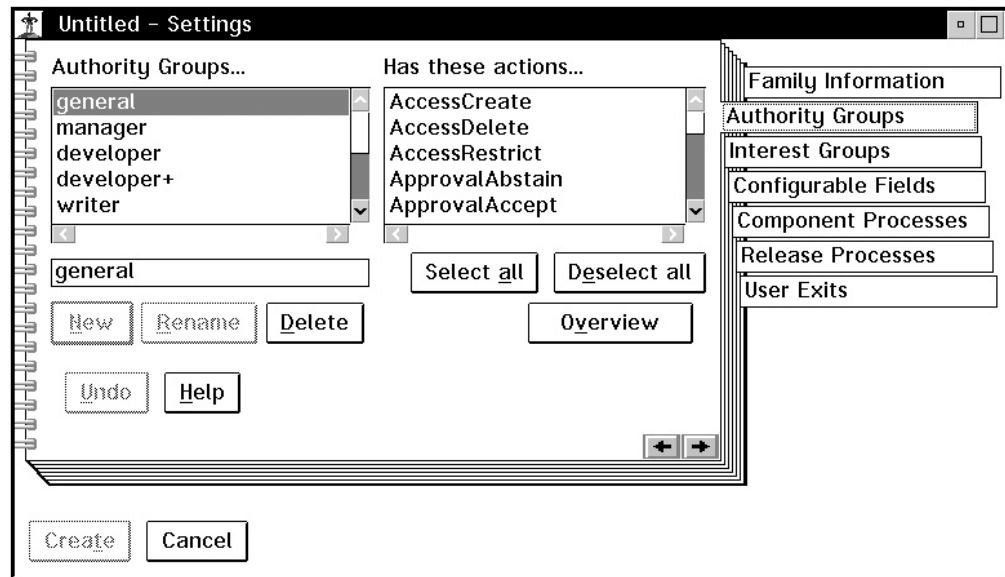


Figure 18. The Authority Groups Settings page

4. Do one of the following:
  - To change an existing group, highlight the group from the **Authority Groups** list, and then select or deselect the appropriate actions from the **Has these actions** list.
  - To create a new group, type the new name in the entry field and select **New**. Then select or deselect the appropriate actions from the **Has these actions** list.

When you type a new name, the actions for the highlighted authority group are highlighted in the **Has these actions** list. Therefore, you might want to highlight an authority group that contains actions similar to the group that you are creating. You will then have fewer actions to select or deselect. You can also select **Deselect all** to deselect all the actions.

To see all the actions that are in a particular authority group, highlight the authority group, and then select **Overview**.

Select **Help** for more information about using the Authority Groups page.
5. When you finish making changes to your notebook pages, select **Update** (or **Create**) to save your changes and exit from the notebook.

The changes will not take effect until you start the family server.

---

## Granting authority to users

Each component has an access list that controls access to development data. Each entry in an access list contains a user ID, the name of an authority group, and whether the authority is granted or restricted for that access group. A user whose user ID appears in the component's access list either has authority to perform any action or is restricted from performing any action listed in the specified authority group. A user ID can appear in a component's access list more than once.

There are three actions you can perform on the entries on a component's access list:

- Add a new entry to the access list. This action grants a user a certain level of authority to access the component.
- Add a new restricted entry to the access list. This action blocks a user from a certain level of authority to access the component.
- Remove an entry from the access list. This action removes a user's granted or restricted authority to access the component. This action can have one of the following effects:
  - If the user has only one entry in the component's access list and has no inherited access to the component, then all access is rescinded.
  - If the user has another entry in the access list or has inherited access to the component, then that user's access is controlled by his or her other entries in the list.

Restricting a user from an authority group is useful when a user has inherited access that you want to rescind. If user ID doug, for example, has releaselead access to component optics, and if component base\_h is a child of component optics, then doug also has inherited releaselead access to component base\_h. TeamConnection allows you to restrict doug's releaselead access to base\_h, so that doug no longer has that level of access to the component. You can also create another access list entry for doug to grant him a lower level of access, developer, for example, to base\_h. Such actions would result in an access list as follows:

Component	User ID	Authority	Type
base_h	doug	releaselead	restricted
base_h	doug	developer	granted

If you do not know what authority groups your organization uses, you can either display the groups on the Show Authority Actions window on the GUI, or ask your family administrator. Do the following from a client machine to display your authority groups:

1. Select **Lists → Access lists → Show authority actions** from the Actions pull-down menu on the Tasks window. The Show Authority Actions window appears.

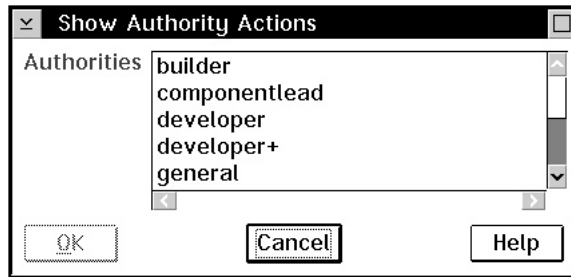


Figure 19. Show Authority Actions window

2. To see a list of the actions that are contained in a group, highlight one or more group names, and then select **OK**.

Before you grant access authority to users, you should understand the following:

- Each component has only one access list.
- The authority groups in the access list must exist in the database.
- Each entry on an access list grants or restricts one user's authority to perform the actions in the specified authority group for the development data managed by that component.
- The authority granted on an access list also grants the specified authority to the user for any descendant components unless the authority has been explicitly restricted from any of those components. Authority is restricted only for that component, not any of its descendant components.
- The total authority a user has is based on the combination of the different authority groups that are associated with the user. For example, a user that has been granted `developer+` authority can perform all the actions listed in that group.

You can create new authority groups that will build on existing groups. For example, you might create a group called `creator` that contains two actions: `compCreate` and `releaseCreate`. You could then grant certain users `creator` authority that would give them this additional authority without duplicating actions that are in their other groups.

- Only the following users can grant or restrict access authority:
  - A superuser
  - The component owner
  - Users with `accessCreate` or `accessRestrict` authority
- You cannot grant authority greater than the authority that you have for a component. For example, if you have `releaselead` authority for the component `optics`, you cannot grant `componentlead` authority to another user. However, if you had `componentlead` authority, you could grant that same authority to another user.
- A user with superuser authority can grant any authority to any user on any access list.

## Granting or restricting access

To grant or restrict access, do one of the following from a client machine:

- From the GUI:

1. Select **Lists → Access lists** from the Actions pull-down menu on the Tasks window, and then select either **Add** or **Restrict**. The Add Access or Restrict Access window appears.

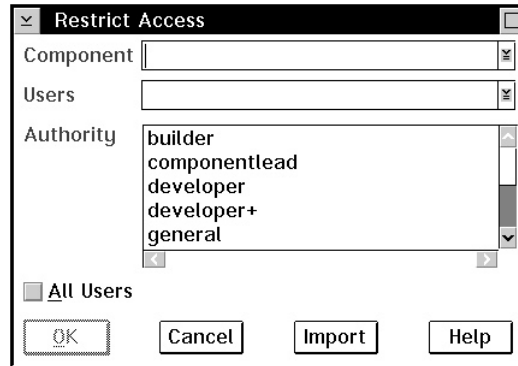


Figure 20. Restrict Access window

2. Type the name of the component and the IDs of the users, and then select the authority group that you want to add them to or restrict them from.  
For more information, select **Help** from the Add Access or Restrict Access window.
  3. Select **OK** to perform the action and close the window, or select **Apply** to perform the action and leave the window open (for the Add Access window).
- From the command line:
    - Use the access -create command to grant authority.
    - Use the access -restrict command to restrict authority.

For example, to give writer authority to a user with an ID of bruce for component robot\_dev, type the following command:

```
teamc access -create -login bruce -authority writer -component robot_dev
```

For more information about the access command, refer to the *Commands Reference* book.

## Removing an entry from an access list

To remove an entry from an access list, do one of the following from a client machine:

- From the GUI:
  1. Select **Lists → Access lists** from the Actions pull-down menu on the Tasks window, and then select **Remove**. The Remove Access window appears.

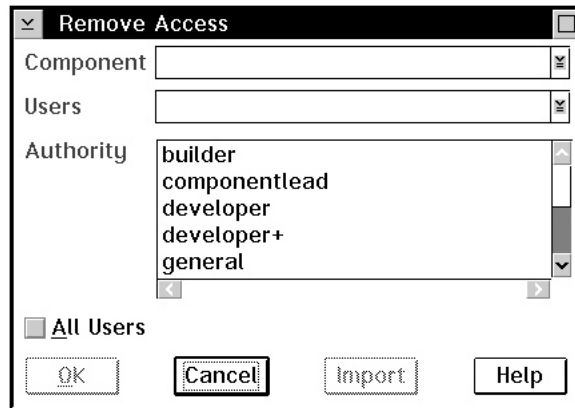


Figure 21. Remove Access window

2. Type the name of the component, the IDs of the users, and then select the authority group that you want to remove from the access list.  
For more information, select **Help** from the Remove Access window.
3. Select **OK** to perform the action and close the window.

- From the command line:

Use the access -delete command to remove an entry from an access list. For example, to remove the entry that grants writer authority to a user with an ID of bruce for component robot\_dev, type the following command:

```
teamc access -delete -login bruce -authority writer -component robot_dev
```

---

## Planning for user notification

Upon request, TeamConnection notifies users when certain actions are performed on certain objects. Notification messages are sent to an electronic mailing address that is specified when the user's ID is created. (See "Setting up the mail facility" on page 85 for more information.)

Some notification is automatic. For example, a user receives notification when someone adds the user's ID to an access list.

Users can receive additional notification. For example, a manager might want to be notified whenever a defect is opened against a component. The component owner can explicitly request that TeamConnection send the manager notification.

Each component has a *notification list* that controls who is notified of what actions. Notification is inherited by descendant components. When a user is to be notified that a specific action occurred within one component, that user will also be notified when that action occurs in any of the descendant components. Unlike authority, notification cannot be restricted for a specific component.

## What are interest groups?

There are many actions that users can be notified of. It would be tedious to request one action notification at a time for each of your users. Instead, you can request

that a user receive notification for a group of actions, called an *interest group*. Each interest group is a group of actions that a certain type of user might want to be notified of.

For example, a developer might want to be notified when defects are opened or closed, while the lead developer needs to be notified not only when defects are opened, modified, or closed, but also when defects are sized or verified.

When planning for notification, you need to be familiar with what type of user is automatically notified when specific actions occur. This information is listed in a table in “Appendix F. Authority and notification for TeamConnection actions” on page 457 . Interest groups are composed of a subset of these TeamConnection actions.

You, the family administrator, are responsible for the interest groups that your organization uses. IBM ships a set of default interest groups with TeamConnection. Determine whether these meet your needs or whether you need to change them. As your organization grows and as your needs change, you will probably want to revise your interest groups.

---

## Creating or modifying interest groups

When the family database is initially created, the interest table contains the default values for the interest groups. If you find that the default interest groups are not adequate, you can create new interest groups or modify existing groups. Interest groups can be created or modified at any time during your development cycle.

First, decide what group of actions the intended users want to be notified of. See if there is a shipped interest group that closely matches your needs. If there is, you might want to modify that group. Otherwise, you will need to create a new group. To help you keep track of the interest groups that the family uses, add the name of each interest group you create to the worksheet provided in “Appendix H. Worksheets” on page 479.

We recommend you use the Family Administrator GUI to create or modify interest groups; however, if you prefer to do this manually, see page 372. Instructions for using the GUI follow.

Go to “Creating or modifying interest groups” on page 372 to complete this task.

Follow these steps to create or modify interest groups from the Family Administrator GUI. Before you do this task, we recommend that you stop the family server (see page “Stopping the servers” on page 95 for instructions).

1. Do one of the following from a server machine to display the TeamConnection Family Administrator window:
  - From the TeamConnection Group folder on the desktop, double-click on the **Family Administrator** icon.
  - Type `tcadmin` from a command prompt.
2. Display the family icon’s pop-up menu, then select **Settings**. The Settings notebook appears.
3. Select the **Interest Groups** page to display the interest group settings.

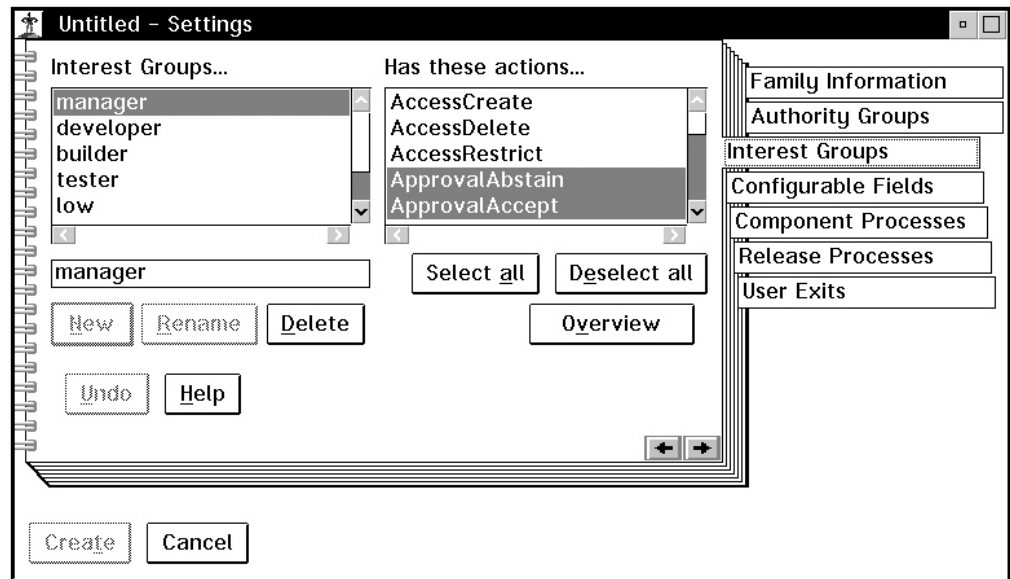


Figure 22. The Interest Groups Settings page

4. Do one of the following:

- To change an existing group, highlight the group from the **Interest Groups** list, and then select or deselect the appropriate actions from the **Has these actions** list.
- To create a new group, type the new name in the entry field and select **New**. Then select or deselect the appropriate actions from the **Has these actions** list.

When you type a new name, the actions for the highlighted interest group are highlighted in the **Has these actions** list. Therefore, you might want to highlight an interest group that contains actions similar to the group that you are creating. You will then have fewer actions to select or deselect. You can also select **Deselect all** to deselect all the actions.

To see all the actions that are in a particular interest group, highlight the interest group, and then select **Overview**.

Select **Help** for more information about using the Interest Groups page.

5. When you finish making changes to your notebook pages, select **Update** (or **Create**) to save your changes and exit from the notebook.

The changes will not take effect until you start the family server.

## Working with notification lists

Each component has a notification list that controls who gets notified of what actions. Each entry in a notification list contains a user ID and the name of an interest group. An interest group defines the actions that each user in the group is to be notified of.

Before working with notification lists, you should understand the following:

- Each component has only one notification list.
- The interest groups listed in the notification list must exist in the database.



- The total notification a user has is based on the combination of the different interest groups that are associated with the user. For example, a user that has been granted med notification will receive notification on actions listed only in that group.  
You can create new interest groups that build on existing groups. For example, you might create a group called size that contains two actions: defectSize and featureSize. You could then add certain users to the size group for a component to give them this additional notification without duplicating actions that are in their other groups.
- Each entry on a notification list ensures that the user will be notified when those actions in the specified interest group occur.
- The user receives notification when actions in the specified interest group occur in any descendant components.
- You cannot restrict notification as you can access authority.

## Displaying interest groups

Before adding users to notification lists, you need to know what interest groups your organization uses. To see a list of groups, do the following from a client machine to display your interest groups on the GUI:

1. Select **Lists → Notification lists → Show interest actions** from the Actions pull-down menu on the Tasks window. The Show Interest Actions window appears.

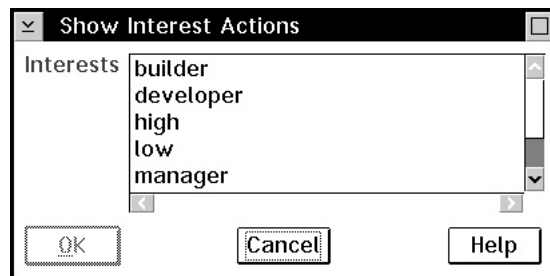


Figure 23. Show Interest Actions window

2. To see a list of the actions that are contained in a group, highlight one or more group names, and then select **OK**.

## Adding an entry to a notification list

To add an entry to a component's notification list, do one of the following from a client machine:

- From the user interface:
  1. Select **Lists → Notification lists → Add** from the Actions pull-down menu on the Tasks window. The Add Notification window appears.

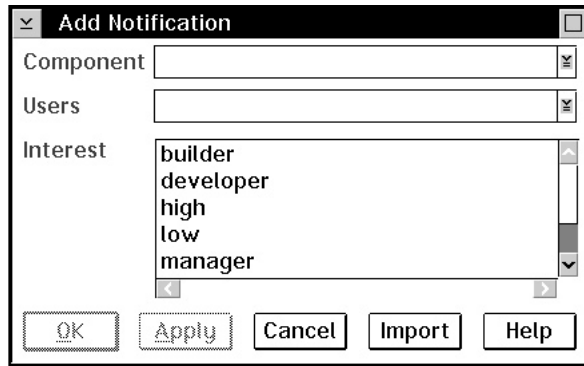


Figure 24. Add Notification window

2. Type the name of the component and the IDs of the users, and then select the interest group that you want to add them to.  
For more information, select **Help** from the Add Notification window.
3. Select **OK** to exit the window or select **Apply** when you want to add the users to another interest group immediately.
- From a command line, use the notify -create command. For example, to add notification to the developer interest group in the robot\_dev component for the owners of user IDs korn and kotora, type the following:

```
teamc notify -create -login korn kotora -interest developer
             -component robot_dev
```

For more information about the notify command, refer to the *Commands Reference* book.

## Removing an entry from a notification list

To remove an entry from a component's notification list, do one of the following from a client machine:

- From the user interface:
  1. Select **Lists** → **Notification lists** → **Remove** from the Actions pull-down menu on the Tasks window. The Remove Notification window appears.

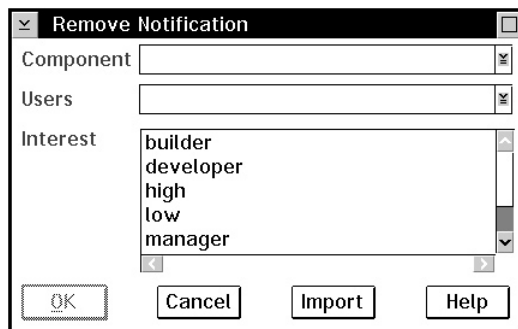


Figure 25. Remove Notification window

2. Type the name of the component and the IDs of the users, and then select the interest group that you want to remove them from.

For more information, select **Help** from the Remove Notification window.

3. Select **OK** to perform the action and close the window or select **Apply** to perform the action and leave the window open.
- From a command line, use the `notify -delete` command. For example, to remove notification from the developer interest group in the `robot_dev` component for the owners of user IDs `korn` and `kotora`, type the following:

```
teamc notify -delete -login korn kotora -interest developer  
-component robot_dev
```



## Chapter 13. Working with configurable fields

Many of the attributes for defects and features are configurable. TeamConnection allows you to customize them so that they more closely match the needs of your development environment. Some examples of attributes that you can customize include the following:

### Defects

Prefix, phase found, phase inject, priority, symptom, target

### Features

Prefix, priority, target

“Appendix B. Configurable field types” on page 387 contains complete lists of attributes that you can customize. These are referred to as configurable fields.

You can also create and add your own configurable fields to defects, features, parts, and users. For example, you might want to add a field called PublImpact to defects and features. Programmers can then use this field to notify the writing team as to whether or not a defect or feature affects the accuracy of the product documentation.

TeamConnection requires two types of objects to make configurable fields work: configurable field types and configurable fields. A *configurable field type* defines possible values for a field, the default value, and a description of each value. The configurable field type, priority, for example, which is shipped with TeamConnection, is defined as follows. This configurable field type is used to define the values for the priority field for features and defects.

Table 19. Definition of configurable field type priority

Possible values	Description
mustfix	Defect or feature must be resolved in this release
candidate	Defect or feature is a candidate if time permits
deferred	Defect or feature deferred to next release
easy	Defect or feature is easy to solve or implement
moderate	Defect or feature is moderately difficult to resolve
difficult	Defect or feature is difficult to solve or implement
n/a	Priority does not apply to this defect or feature

A *configurable field* defines how the configurable field type is to be used by the object for which it is defined. A configurable field definition includes the following information:

- Whether or not the field is active
- Whether it is required or optional
- Whether you can set its value on Create or Open windows, or just modify it on Modify Property windows

- The configurable field type that defines the values and default for the field
- The attribute name to be used for the field on TeamConnection commands
- The label to be used for the field on Create, Open, and Modify Property windows
- The title to be used for the field in reports and in Features, Defects, Parts, and Users windows

Features and defects use the priority field type differently, for example, and the difference between the two is determined by how the configurable field is defined. The following table shows how options set for priority differ for features and defects.

*Table 20. Definition of priority field in features and defects*

Option	Features	Defects
Active	Yes	Yes
Required	No	No
Allow on Create/Open	No	Yes
Field Type	priority	priority
Attribute	priority	priority
Field Label	Priority	Priority
Title	Priority	Priority

Before you configure new fields, you must decide what values will be acceptable entries for the fields. To do this, you specify a configurable field type when you create the field. If you do not want to use an existing field type, you must create the type before you create the field.

For example, if you create a PublImpact field, you might want to create a new configurable field type called PubImpact (the configurable field type can have the same name as the field). If you assign the attributes of yes, no, and maybe to this field, writers can access the user interface or issue a command to get a list of all defects or features that affect the publications. If you also add an attribute of done, the writers can indicate when they have finished updating the documentation.

---

## Defining configurable field types

For the defect and feature tables, TeamConnection ships configurable field types that have defined values. You can use the default field types as is or change their attributes. For example, TeamConnection defines a field named Severity. This field has valid values of 1, 2, 3, and 4 (see the table on page 387). You could add an additional value of 5, or you could change the description of what the value 2 represents.

**Note:** The maximum size for the value of a configurable field type is 85 characters (single-byte or double-byte).

You can also create new configurable field types for the defect, feature, part, and user tables. This allows you to structure problem tracking information for your development environment. You can create new types or change the attributes of the existing types at any time during your development cycle. TeamConnection stores configurable field types in a file called config.ld. “Defining configurable field types” on page 375 describes the config.ld file.

It is recommended that you use the Family Administrator GUI to create or modify configuration field types; however, if you prefer to do this manually, see page 375. Always backup your database before adding or changing configurable fields or configurable field types. Instructions for using the GUI follow.



For UNIX environments, use the methods described in “Defining configurable field types” on page 375 to complete this task.

Follow these steps to create or modify configurable field types from the Family Administrator GUI. Before you do this task, stop the family server (refer to page “Stopping the servers” on page 95 for instructions).

1. Do one of the following from a server machine to display the TeamConnection Family Administrator window:
  - From the TeamConnection Group folder on the desktop, double-click on the **Family Administrator** icon.
  - Type `tcadmin` from a prompt.
2. Display the family icon’s pop-up menu; then select **Settings**. The Settings notebook appears.
3. Select the **Configurable Fields** page to display the current settings.

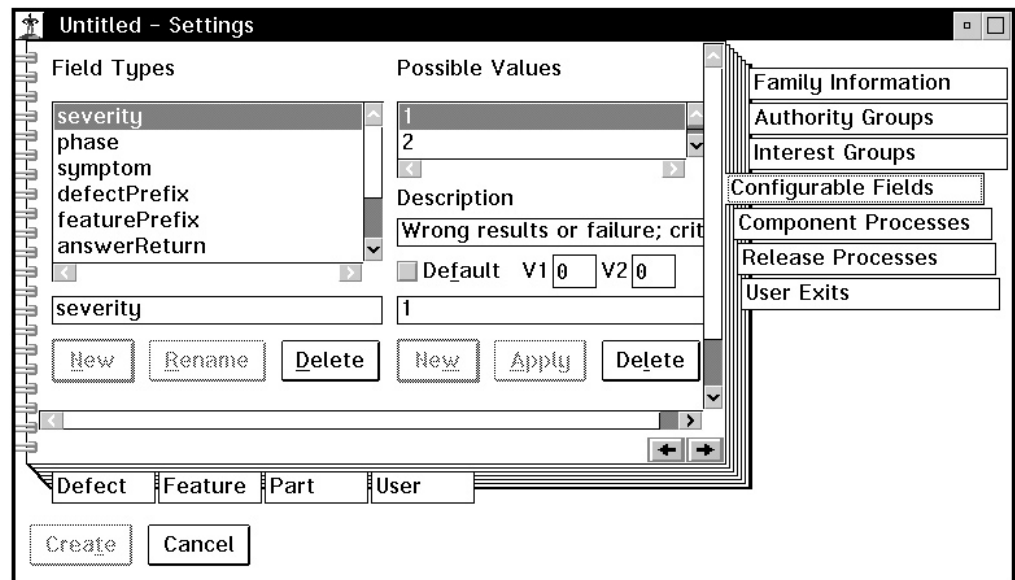


Figure 26. The Configurable Fields Settings page

From this page, you can do the following:

- To change an existing field type, highlight it in the **Field Types** list. Information about the selected type is displayed to the right of the list, such as possible values and description. You can then change this information.
- To create a new field type, follow these steps:

- a. Type the new name in the entry field below the **Field Types** list and select the **New** push button below the entry field.
- b. To define values for the new field type, type a value in the entry field below the **Possible Values** list, type a description for the value in the **Description** field, and then select the **New** push button below the entry field. Repeat this step for each possible value for the configurable field. Configurable field values cannot exceed 85 characters and cannot contain spaces.
- c. To set one of the values you have defined as the default, select the value in the **Possible Values** list, select the **Default** checkbox, and then select the **Apply** push button.

**Note:** The **V1** and **V2** fields to the right of the **Default** checkbox are not used.

For an explanation of how the configurable field type options on the Configurable Fields page of the settings notebook, in the config.ld file, and in the configurable field types table correspond, see “Appendix B. Configurable field types” on page 387.

- To rename or delete an existing field type, type the name in the entry field, and then select **Rename** or **Delete**.

Select **Help** for more information about using this Settings page.

4. When you finish making changes to your notebook pages, select **Update** (or **Create**) to save your changes and close the notebook.

---

## Changing or creating configurable fields

Default configurable fields are shipped by IBM and are installed when the TeamConnection server is configured. If you do not want to use these defaults, you can change them at any time after the family database is created.

The following conditions apply to the use of configurable fields:

- Part, and user objects can have up to 20 configurable fields each. Defect and feature objects can have a combined total of 20 configurable fields.
- Configurable fields cannot be deleted; however, they can be renamed.
- Fields for defect and feature objects are effective on open and modify actions. Fields for part and user objects are effective on create and modify actions.
- You can use the data from configurable fields to search the database and display information in reports, but TeamConnection does not use the data. For example, if you have a field called PubImpact, TeamConnection cannot change the state of a defect based on the value of this field, but users can sort all defects and features by whether or not they impact the publications.
- When you add fields, TeamConnection displays them on the GUI like any predefined field. However, the help information for configurable fields for the GUI and the commands do not reflect your new or changed fields.
- Whenever you create or modify a configurable field, your client users need to do one of the following to make the new field appear on the GUI:
  - Close the GUI and reopen it
  - Use the settings notebook to change the family and then change it back

Refreshing the GUI in this way is particularly important if the new field is required. Otherwise, your users will receive errors, but will have no way to enter information in the required fields.



- The data type for all configurable fields is character. The value for configurable fields cannot exceed 85 characters and cannot contain spaces.

## Creating and modifying configurable fields

We recommend you use the Family Administrator GUI to create or modify configurable fields; however, if you prefer to do this manually, see page 378. Instructions for using the GUI follow.



Go to “Creating and modifying configurable fields” on page 378 to complete this task.

Follow these steps to create or modify configurable fields from the Family Administrator GUI. Before you do this task, stop the family server (refer to page “Stopping the servers” on page 95 for instructions).

1. Do one of the following from a server machine to display the TeamConnection Family Administrator window:
  - From the TeamConnection Group folder on the desktop, double-click on the **Family Administrator** icon.
  - Type tcadmin from a prompt.
2. Display the family icon’s pop-up menu; then select **Settings**. The Settings notebook appears.
3. Select the **Configurable Fields** page.  
This page has the following tabs: Defect, Feature, Part, and User. Select one of the tabs to display the configurable field settings for that object.

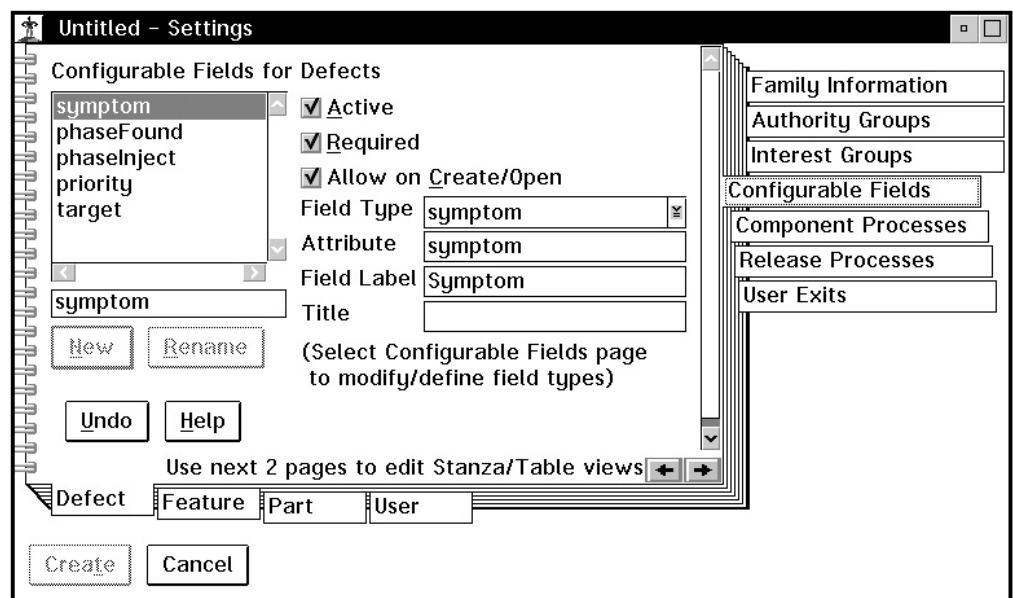


Figure 27. The Configurable Fields for Defects Settings page

From this page, you can do the following:

- To change information about a field, highlight the field from the **Configurable Fields** list. Information about the selected field is displayed to the right of the list. You can then change this information.
- To create a new configurable field, follow these steps:
  - a. Type the new name in the entry field below the **Configurable Fields** list and select **New** to add the field name to the list.
  - b. Set options for the new configurable field by selecting or entering information in the fields to the right of the list.

**Active** Select this checkbox to activate the new configurable field. If this checkbox is not selected, the field does not appear on the GUI.

**Required**

Select this checkbox to make the new field required on Create and Open windows. If this checkbox is not selected the field is optional. This checkbox is not available for Parts.

**Note:** A configurable field that is required, and has a field type with a default value set behaves like an optional field because if no value is given, a default has already been specified.

**Allow on Create/Open**

Select this checkbox to add the field to the Create or Open windows for defects, features, or users. If this checkbox is not selected the field appears only on the Modify Properties windows. This option is not available for parts.

**Field Type**

Select a predefined configurable field type from this list. This field creates a pointer to the entries in config.ld that define the configurable field type you enter here. TeamConnection retrieves the possible values for the configurable field from config.ld. See "Defining configurable field types" on page 142 for instructions on creating a configurable field type.

**Attribute**

Type the name to be used for this field on TeamConnection line commands.

**Field Label**

Type the name to be used for this field on the Create, Open, Filter, or Modify Properties window.

**Title** Type the name to be used for this field on the Defects, Features, Parts, or Users window.

For an explanation of how the configurable field options on the settings notebook, in the chfield command, and in the features, defects, parts, and users tables correspond, see "Appendix B. Configurable field types" on page 387 .

- To rename an existing field, type the name in the entry field and then select **Rename**. If you rename the field and want to change the possible values for the field, type the appropriate information to the right of the list, and then select **Update** (or **Create**).

Select **Help** for more information about using this Settings page.

As noted earlier, the number of configurable fields that you can have is limited. You cannot delete configurable fields, but you can rename them and change their properties.

4. When you finish making changes to your notebook pages, select **Update** (or **Create**) to save your changes and exit from the notebook.

The changes do not take effect until you start the family server.

## Displaying configurable field properties

To display the properties of the active configurable fields for an object, type one of the following commands from a prompt:

- `teamc defect -configInfo -family familyName [-raw]`
- `teamc feature -configInfo -family familyName [-raw]`
- `teamc part -configInfo -family familyName [-raw]`
- `teamc user -configInfo -family familyName [-raw]`

These commands show you exactly what has been defined and let you verify that the fields were loaded correctly.

If the `-raw` flag is used, the information is organized in a fixed ASCII table format as follows:

```
Field Label|Title|Attribute|DB Column Name|Create|Required|Field Type
```

**Note:** The properties of both the Prefix and Severity configurable fields are displayed for defects, whereas only the Prefix field is displayed for features.

---

## Changing report formats

TeamConnection users can view or print reports about an object. When you create a field, TeamConnection adds the new field to the report. You can choose the field information that you want to present to the user and the place on the report where the information appears.

Reports are displayed in two formats: stanza and table. Continue reading for a description of these formats and an explanation of how you can change them.

### The stanza report

Figure 28 on page 148 shows an example of a stanza report. Each line in the report represents one or more attributes of the object. To display the report from a command prompt, type the following command. This example displays information about defect 3168.

```
teamc report -view DefectView -where "name='3168'" -stanza
```

From the GUI, select **Defects** → **View** from the Actions pull-down menu. When the View Defect Information window appears, type the defect name.

The following is an example of a stanza report:

prefix	d		
name	3168		
reference	testcase_099		
abstract	Compilation error occurred.		
duplicate			
state	open	priority	
severity	2	target	driver_020
age	9		
compName	demoComponent	answer	
release	demoRelease	symptom	compile_failed
envName		phaseFound	prototyping
level	level_019	phaseInject	
addDate	93/04/01 11:32:47	assignDate	93/04/04 18:45:41
lastUpdate	93/04/13 11:54:15	responseDate	93/04/03 11:29:59
endDate			
ownerLogin	annHar	originLogin	martin
ownerName	Ann Harrison	originName	Martin Karland
ownerArea	Development	originArea	Testing
developer	johnDoe		

Figure 28. Sample stanza report displayed after adding configurable fields

We recommend you use the Family Administrator GUI to change the stanza report formats; however, if you prefer to do this manually, see page 381. Instructions for using the GUI follow.



Go to “Changing report formats” on page 381 to complete this task.

Follow these steps to change the position of the field, or to change or delete the format specification from the Family Administrator GUI. Before you do this task, stop the family server (refer to page “Stopping the servers” on page 95 for instructions).

1. Do one of the following from a server machine to display the TeamConnection Family Administrator window:
  - From the TeamConnection Group folder on the desktop, double-click on the **Family Administrator** icon.
  - Type `tcadmin` from a prompt.
2. Display the family icon’s pop-up menu; then select **Settings**. The Settings notebook appears.
3. Select the Configurable Fields page.

This page has the following tabs: Defect, Feature, Part, and User. Select one of the tabs to display the configurable field settings for the selected object. Select the arrow at the bottom right corner of the page to display the Stanza View Format page for that object.

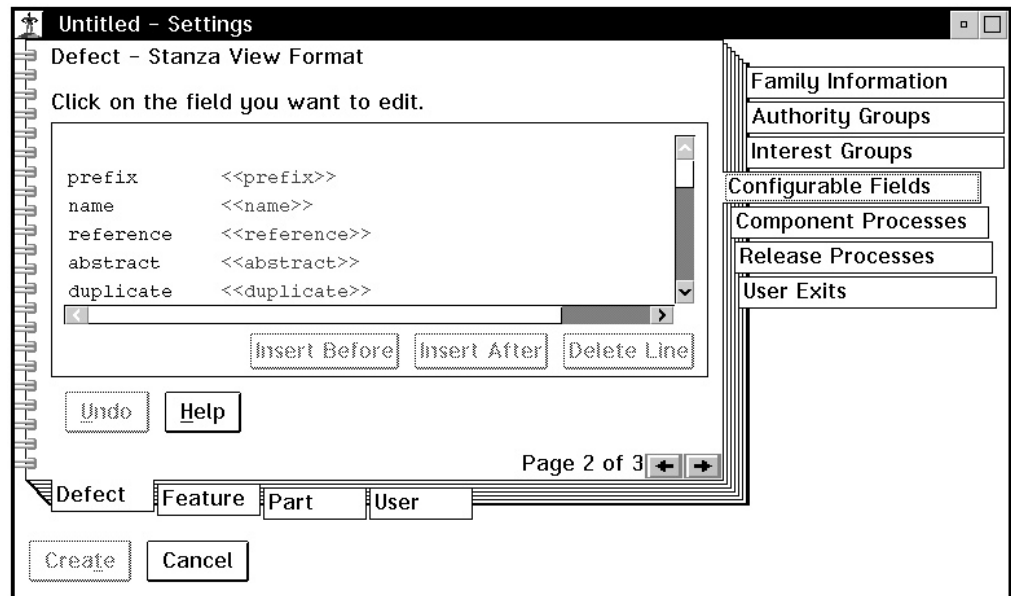


Figure 29. The Stanza View Format Settings page

4. Do one of the following:
  - The first and third columns are input fields. Type in these fields to change the report labels.
  - The second and fourth columns specify the values you want displayed. To change the value for a specific field:
    - a. Highlight the field.
    - b. Press the right mouse button to display a list of the acceptable values.
    - c. Select the value you want to use.
  - Use the **Insert Before** and **Insert After** push buttons to add blank lines into which you can insert new fields.
  - Use the **Delete Line** push button to remove a field from the report. This button removes all fields on the selected line. If the line defines fields in columns one and three, then both fields are deleted.
 

To delete a field defined in columns three and four, select the value in column four with mouse button 2 and then select <<EMPTY>>. Then select the value in column three with mouse button 1 and delete the text.
  - To add a previously deleted field back to the report, follow these steps:
    - a. Select a blank line.
    - b. Press mouse button 2 in column 2 or 4 to display a list of fields you can add to the report, then select a field from this list. This action adds the field to column two or four of the blank line.
    - c. Place the mouse pointer in the blank column beside the field you just added and type a field name on column one or three.
5. When you finish making changes to your notebook pages, select **Update** (or **Create**) to save your changes and exit from the notebook.

**Note:** When changing the Part stanza format, only the partView table is changed, not the partFullView table. To manually change the partFullView table, see page 381.

Refer to the *Commands Reference* if you are not familiar with the differences between these two views.

The changes do not take effect until you start the family server.

## The table report

Figure 30 is an example of a table report. Each row of the table represents a different object, and the value of each attribute for that object is displayed in columns. The following is an example of the command to display the report.

```
teamc report -view DefectView -where "name='3168'" -table
```

The following is an example of the table format for defects. This example shows a table report to which a developer field has been added.

pref	name	compName	state	originLo	ownerLog	sev	age	prio	abstract	developer
d	3168	demoComponent	open	martin	annHar	2	009		Compilation error	johnDoe

Figure 30. Sample table report displayed after adding configurable fields

It is recommended that you use the Family Administrator GUI to change the table formats; however, if you prefer to do this manually, see page 381. Instructions for using the GUI follow.



Go to “Changing report formats” on page 381 to complete this task.

Follow these steps to change the columns you want displayed, their position in the table, or the width of the columns. Before you do this task, stop the family server (refer to page “Stopping the servers” on page 95 for instructions).

1. Do one of the following from a server machine to display the TeamConnection Family Administrator window:
  - From the TeamConnection Group folder on the desktop, double-click on the **Family Administrator** icon.
  - Type `tcadmin` from a prompt.
2. Display the family icon’s pop-up menu; then select **Settings**. The Settings notebook appears.
3. Select the Configurable Fields page.

This page has the following tabs: Defect, Feature, Part, and User. Select one of the tabs to display the configurable field settings for that object. Go to page 3 by twice selecting the arrow at the bottom right corner of the page to display the Table View Format page for the selected object.

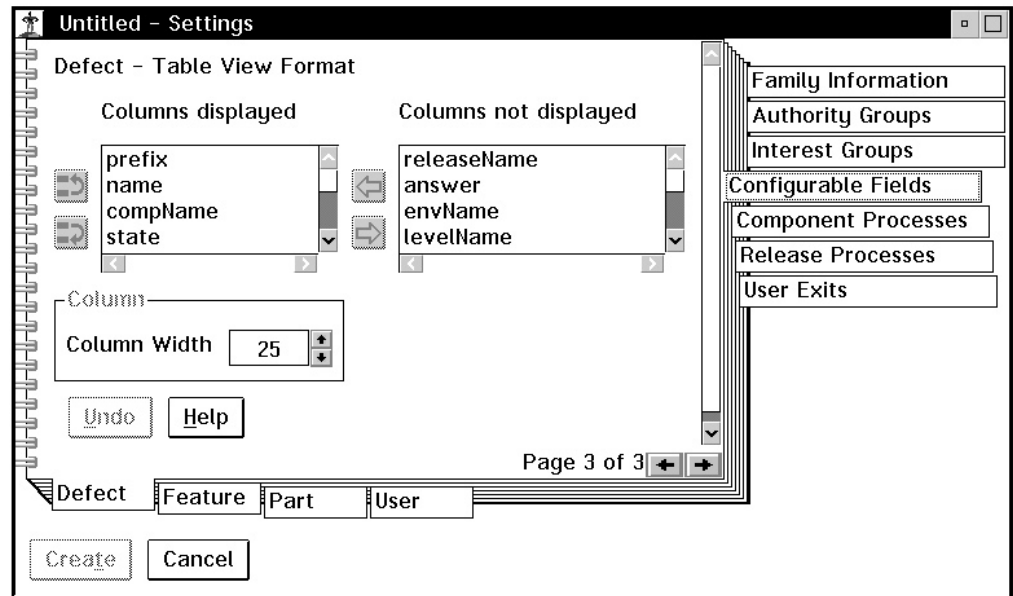


Figure 31. The Table View Format Settings page

4. Do one of the following:
  - To change the position of a column, highlight the column label and then select the up or down arrow repeatedly until the column name is in the correct position.
  - To remove a column from the table, select the column label from the **Columns displayed** field and then select the highlighted right arrow next to the **Columns not displayed**.
  - To add a column to the table, select the column label from the **Columns not displayed** field and then select the highlighted left arrow. The label is placed at the bottom of the list; you can change its position.
  - To change the width of the column, highlight the column label and then select the up or down arrow in the **Column width** field.
5. When you finish making changes to your notebook pages, select **Update** (or **Create**) to save your changes and exit from the notebook.

**Note:** When changing the Part table format, only the partView table is changed, not the partFullView table. To manually change the partFullView table, see page 381.

Refer to the *Commands Reference* if you are not familiar with the differences between these two views.

The changes do not take effect until you start the family server.





---

## Chapter 14. Configuring family processes

TeamConnection is shipped with several predefined processes for both components and releases. If these processes do not meet the needs of your development organization, you can create your own processes. You do this by combining some of the predefined subprocesses that IBM provides.

This chapter explains how you configure new processes or change existing processes. If you are not familiar with TeamConnection processes and how they are used, read “Planning your processes” on page 115.

Consider the following before configuring your processes:

- To avoid confusion for end users, do not modify processes after they are in use. If you must add or delete subprocesses in an existing process, consider instead creating a new process.
- Do not delete a process that is being used by any component or release in your family.
- Changes to processes do not take effect until one of the following occurs:
  - A component or release is modified using the process name
  - A component or release is created using the process name

The processes shipped by IBM are explained to your users in online help. Any processes you configure are not explained in online help. Therefore, you will need to provide this type of information to your users.

Tables are provided in “Configurable processes worksheets” on page 488 for you to record the processes you configure.

---

### Modifying or creating configurable processes

Your first step in configuring a process is to give the process a name. The name you choose can be up to 15 characters in length with no blanks, tabs, or vertical separators. To help you keep track of the processes that you configure for your family, add the name of each process you create to the worksheet provided in “Configurable processes worksheets” on page 488.

We recommend you use the Family Administrator GUI to modify or create configurable processes; however, if you prefer to do this manually, see page 373. Instructions for using the GUI follow.

Follow these steps from a server machine to configure component or release processes from the Family Administrator GUI. Before you do this task, we recommend that you stop the family server (refer to page “Stopping the servers” on page 95 for instructions).

1. Do one of the following to display the TeamConnection Family Administrator window:
  - From the TeamConnection Group folder on the desktop, double-click on the **Family Administrator** icon.
  - Type `tcadmin` from an OS/2 command prompt.
2. Display the family icon’s pop-up menu, then select **Settings**. The Settings notebook appears.

3. Select **Component Processes** or **Release Processes**.

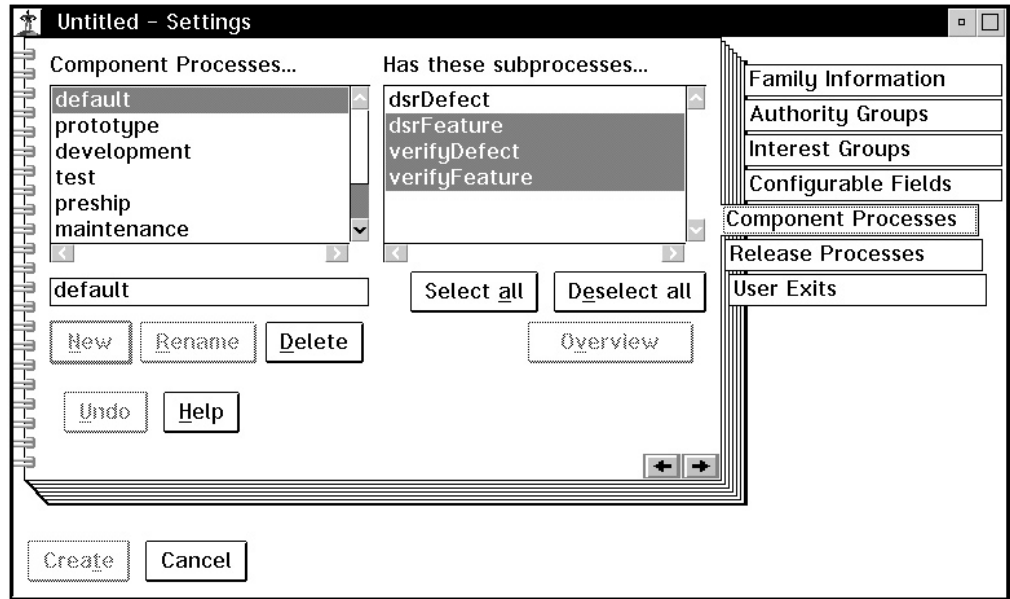


Figure 32. The Component Processes Settings page

4. Do one of the following:
  - To change a process, highlight a process from the list of processes, and then select or deselect the appropriate subprocesses.
  - To create a new process, type the new name in the entry field and select **New**. Then select or deselect the appropriate subprocesses.
5. When you finish making changes to your notebook pages, select **Update** (or **Create**) to save your changes and exit from the notebook.

The changes do not take effect until you start the family server.

---

## Chapter 15. Providing user exits

TeamConnection provides a highly configurable set of processes so that you can adapt the tool to your specific needs. However, there are many cases where you might want to make further process adjustments or add automated steps. User exits allow you to enhance the TeamConnection processes to perform tasks like the following:

- Ensuring that code files meet formatting requirement, such as the inclusion of keywords that identify the name and version of the file
- Creating a new defect when a verification record is rejected on a current defect
- Analyzing a build failure and removing any work areas from a driver that include changes to files that failed to compile (if the build fails the driver remains in restrict state, however if it succeeds the driver can be committed)
- Ensuring that the right information is in a sizing record before it is accepted
- Automatically generating management reports when a driver is committed

This chapter describes user exits, how to use them, and how to implement them for each TeamConnection family. User exits are not necessary for the operation of TeamConnection; they are optional and can be configured for each family.

With user exits, you can specify additional actions that you want performed before completing or proceeding with a specific TeamConnection command action. A user exit enables the TeamConnection server to call a user-defined program during the processing of TeamConnection actions. The program can be an executable file or a command file. Thus, you can use TeamConnection as a trigger to start non-TeamConnection processing. You can also use user exits to restrict certain TeamConnection actions based on external considerations. For example, you might have a user exit scan C source files to ensure that the source code conforms to the standards defined by your development process.

The userExit file indicates the programs you want started for specific TeamConnection actions. You can add entries to this file using the Family Administrator GUI, which is the recommended method, or directly edit the file. This chapter explains how to set up user exits using the Family Administrator GUI. For instructions on editing the userExit file directly, see “Setting up user exits” on page 384 .

**Note:** The userExit file is copied to your family database directory from a file located in the language subdirectory of the nls\cfg directory path in the TeamConnection installation directory, for example, teamc\nls\cfg\enu. The version of the userExit file in this location contains comments that are not copied when the family is created using the Family Administrator GUI.

User exits are provided for most TeamConnection actions. The actions that support user exits are listed in “Appendix C. User exit parameters” on page 393.

---

### Writing user exit programs

User exit programs have the following behavior:

- You can call an exit at the following times:

For the user exit program to...	Select this if using the GUI;	Specify this if manually updating the userExit file
Start at the beginning of the action, before any initialization or access checking takes place.	Before action checking	Exit ID 0
Start after all checks are made and TeamConnection is ready to process the command.	Before actions processing	Exit ID 1
Start after action is completed and all database or library updates have been committed.	After action completes	Exit ID 2
Start when a previous user exit with an exit ID of 0 or 1 is not successful, or when the action is not successful. This user exit program can clean up what the other user exit programs started.	If action or programs fail	Exit ID 3

- Each user exit program receives a unique list of parameters, defined as follows:

#### **UEprogram**

Name of the calling user exit.

#### **UEparameter**

The user-defined parameter in the userExit file.

#### **Action parameters**

The parameters passed to the user exit program from the TeamConnection command. The first parameter is the name of an environment file that contains the name and value of any action parameters specified by the family administrator.

- When a user does not enter a value in an optional GUI field or command line, the user exit puts "" in its place. This is also true for UEparameters.
- Positional parameters that pass true or false values, such as **Break common link** (force flag), return the following:

**True** 1

**False** 0

- A nonzero return code from a user exit program for exit ID 0 or 1 terminates the TeamConnection command. Nonzero return codes do not affect exit ID 2 or 3.
- The userExit file is read only when the family server is started. After a user exit is enabled, you can change the exit without restarting the family server. However, you must stop and restart the server before your changes are recognized.

Follow these guidelines when you write user exit programs:

- Limit the length of time that the user exit program runs.
- Not all TeamConnection commands can be used in a user exit. Some may cause a database deadlock to occur.
- User exit programs do not permit user interaction (for example, from a user exit program, you cannot prompt a user with a read command).
- Define only one user exit program for each TeamConnection action and exit ID combination. If you define more than one program, TeamConnection uses the last one you define.

- You are limited to a total of 40,000 bytes of total output from all user exits, plus warnings, for each TeamConnection action (except teamc report and -view actions).

The Family Administrator GUI guides you through the configuring of user exits, and the following examples give you a start on writing your own user exits:

- viewexit.c: A c program that displays the output of a user exit, identifying the user exit command, the environment file, the user-defined parameter, and each of the positional parameters. Also, parameters that are null or were truncated because they would make the command string too long are identified. There is also a function that was derived from teamcenv (below) to display the contents of the environment file (if one was specified).
- teamcenv.c: A c program that lets you read the entire environment file and display the contents of each variable stored, or extract the value of a specific variable based on the name. It can be called from viewexit.c.
- viewexit.ksh: A version of viewexit.c written in Korn shell.
- viewexit.cmd: A version of viewexit.c written in REXX.

The table in “Appendix C. User exit parameters” on page 393 lists the parameters that are passed by each TeamConnection action. The table also contains descriptions of many of the parameters.

The following table provides examples of the parameters passed to a user exit program for the PartAdd action. Each parameter has a numeric identifier. The first column shows how to identify the parameter in source for an executable (such as C). The second column shows how to identify the parameter in source for a shell program (such as an OS/2 .cmd file or an AIX .ksh file). The third column describes each parameter.

*Table 21. Parameters passed to a user exit for the PartAdd action*

Argument number in an executable	Argument number in a shell program	Argument value
Exit ID 0		
arg[0]	\$0	User exit program name
arg[1]	\$1	User defined parameter - required field containing a string passed to a user exit program. The string must be in quotes to ensure that all subsequent parameters are in the correct position in the argument list.
arg[2]	\$2	Environment file name - null if not used
arg[3]	\$3	File path name
arg[4]	\$4	'0' if the file will not be transmitted; otherwise the file will be transmitted.
arg[5]	\$5	Client location
arg[6]	\$6	Temp file on server
arg[7]	\$7	Release name
arg[8]	\$8	Component name
arg[9]	\$9	File type: set to '2' if binary is specified at file creation, to '1' if text, or, otherwise, to '0'

arg[10]	\$10	Work area name
arg[11]	\$11	FileMode
arg[12]	\$12	Parent name
arg[13]	\$13	Parser name
arg[14]	\$13	Builder name
arg[15]	\$15	Parent relation type
arg[16]	\$16	Builder parms
arg[17]	\$17	Part type
arg[18]	\$18	Parent type
arg[19]	\$19	Temporary to build
arg[20]	\$20	Effective TeamConnection user ID (TC_BECOME)
arg[21]	\$21	Real TeamConnection user ID (TC_USER)
arg[22]	\$22	Verbose flag

---

#### Exit ID 1, 2

---

arg[0]	\$0	User exit program name
arg[1]	\$1	User defined parameter - should be in quotes
arg[2]	\$2	Environment file name - null if not used
arg[3]	\$3	File path name
arg[4]	\$4	Temp file on server
arg[5]	\$5	Release name
arg[6]	\$6	Component name
arg[7]	\$7	File type: set to '2' if binary is specified at file creation, to '1' if text, or, otherwise, to '0'
arg[8]	\$8	Work area name
arg[9]	\$9	Remarks
arg[10]	\$10	FileMode
arg[11]	\$11	Parent name
arg[12]	\$12	Parser name
arg[13]	\$13	Builder name
arg[14]	\$14	Parent relation type
arg[15]	\$15	Builder parms
arg[16]	\$16	Part type
arg[17]	\$17	Parent type
arg[18]	\$18	Temporary to build
arg[19]	\$19	Effective TeamConnection user ID
arg[21]	\$21	Verbose flag

---

#### Exit ID 3

---

Same as Exit ID 0 with an additional parameter as the last parameter to indicate the last user ExitID that has been executed successfully, for example, 0 or 1.

---

If you have a user exit program called viewexit, for example, that you want to execute when a user creates a part in TeamConnection, you would include the following line in the userExit file:

```
PartAdd 1 viewexit "1991 1992" # copyright for PartAdd with '1991 1992'
```

This program runs when TeamConnection recognizes that the user is requesting a valid PartAdd action, but before the PartAdd action actually starts. If the content of the viewexit user exit program is:

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>

extern int errno;

/* This is based on a limit used for actions in TeamC */
#define maxParmName 40

/*-----\
envGetFromEnvFile:
- Read each entry to environment file
- Read binary:
  size of parameter, parameter string, size of value, value string
- Minimal error checking to simplify example
Note: This routine is condensed from the sample program teamcenv.c
\-----*/
int envGetFromEnvFile(char *envFileName)
{
    FILE *envFile;

    int nNameLength;
    int nValueLength;
    char parameterName[maxParmName+1]; /* allow for maximum in TeamC (15 + NULL) */
    char parameterValue[16001]; /* allow for max in TeamC 16000 for remarks + NULL */

    /* Open temporary file */
    envFile = fopen(envFileName, "rb");
    if (envFile == NULL)
    {
        fprintf(stderr, "teamcenv: Error, could not open file \"%s\\n\",
            envFileName);
        return 1;
    }

    /* Dump all attributes */
    printf("\
Parameter                                Value\\n\
=====\\n");

    fread(&nNameLength, sizeof(int), 1, envFile);
    fread(parameterName, sizeof(char), nNameLength, envFile);
    *(parameterName+nNameLength)='\0';
    fread(&nValueLength, sizeof(int), 1, envFile);
    fread(parameterValue, sizeof(char), nValueLength, envFile);
    *(parameterValue+nValueLength)='\0';

    while (!feof(envFile))
    {
        strncat(parameterName, "
            (maxParmName - strlen(parameterName)));
        *(parameterName+maxParmName) = '\0';
        printf("%s %s\\n", parameterName, parameterValue);
        fread(&nNameLength, sizeof(int), 1, envFile);
        *(parameterName+nNameLength)='\0';
        fread(parameterName, sizeof(char), nNameLength, envFile);
        fread(&nValueLength, sizeof(int), 1, envFile);
        fread(parameterValue, sizeof(char), nValueLength, envFile);
    }
}
```

```

        *(parameterValue+nValueLength)='\0';
    }

    fclose(envFile);
    return 0;
}

/*-----*/
main:
- Print standard set of arguments at beginning of parameter list:
  UEprogram, UEparameter and EnvFile
- Print rest of arguments; the positional parameters
- If EnvFile is not NULL, print the contents of the EnvFile
/*-----*/
int main(int argc, char *argv[])
{
    int i,n;

    /* Compute the number of action parameters passed at definition */
    int totalParms = argc - 3;

    /* Display the name of the command. */
    /* It is not necessary to parse name. */
    printf("UEProgram:          %s\n", argv[0]);

    /* Display parameter list. */
    if (strlen(argv[1]) != 0)
        printf("UEParameters string:    %s\n", argv[1]);
    else
        printf("UEParameters string is NULL\n");

    /* Display parameter env file. */
    if (strlen(argv[2]) != 0)
        printf("EnvFile name:          %s\n", argv[2]);
    else
        printf("No environment file; string is NULL\n");

    /* Display parameter list. */
    for (i = 3; i < argc; i++)
    {
        n = strlen(argv[i]);
        if (n == 0)
            printf("Action Parameter [%d] is NULL\n", i-2);
        else
        {
            printf("Action Parameter [%d]:    %s\n", i-2, argv[i]);

            /* Ellipses in data, checking for end (i.e. truncation) */
            /* - Each parameter limited to 400 bytes in OS/2, Windows */
            /*   and Windows NT, and 16000 bytes in Unix */
            /* - Total command string limited to 1024 bytes in OS/2, */
            /*   Windows and Windows NT, and 32000 bytes in Unix */
            /* - compute address of last 3 characters then compare */
            if (strcmp((argv[i])+n-3, "...") == 0)
            {
                printf("                parameter %d string was \
truncated!\n", (i-1));
                /* If last parameter truncated, then entire list truncated */
                if (i == (argc - 1))
                    printf("Parameter list was truncated!\n");
            }
        }
    }

    printf("Total action parameters: %d\n", totalParms);

    /* Print contents of parameter env file. */
}

```



```

    if (strlen(argv[2]) != 0)
    {
        printf("Printing contents of Environment File\n");
        envGetFromEnvFile(argv[2]);
    }

    return (0);
}

```

and the command issued by the TeamConnection client is:

```

teamc part -create src\partX.c -component codeAcomp -release ToolAvRel
        -family testfam -workarea waABC -parent partX.obj -input -parser Cparser

```

the output displayed in the client window is:

```

UEProgram:          C:\TESTFAM\BIN\viewexit.cmd
UEParameters string: 1991 1992

No environment file; string is NULL
Action Parameter {1}: src/partX.c
Action Parameter {2}: C:\TESTFAM\BIN\TCTMP\D373\RQ0ESU7J.C2T
Action Parameter {3}: ToolAvRel
Action Parameter {4}: codeAcomp
Action Parameter {5}: 1
Action Parameter {6}: waABC
Action Parameter {7}: Initial Version of src/partX.c
Action Parameter {8}: 0600
Action Parameter {9}: partX.obj
Action Parameter {10}: Cparser
Action Parameter {11}: is NULL
Action Parameter {12}: 1
Action Parameter {13}: is NULL
Action Parameter {14}: TcPart
Action Parameter {15}: TcPart
Action Parameter {16}: no
Action Parameter {17}: cmvctest
Action Parameter {18}: 1
Total action parameters:18

```

In the preceding example, no remark was specified so TeamConnection provided Initial Version of src/partX.c.

---

## Environment file

The list of positional parameters in a user exit command string can get very long. Further, it is sometimes difficult to parse through such a command. For example, in OS/2, a carriage return in a parameter prematurely ends the command string. Therefore, it is not always certain that the command string will include the parameters after a remarks parameter.

The solution to this problem is to select the variables that you will need and have their values inserted into an environment file. The environment file is a binary file that contains the names of parameters and their values, including carriage returns and anything else that has been passed along. If the use of an environment file is specified, the name of the temporary file containing the environment variable names and values is the positional parameter after the user-defined parameter.

The viewexit.c and teamcenv.c files in the samples directory show how to read the environment file.

“Appendix C. User exit parameters” on page 393, containing the names of all of the parameters, is also the list of names used for the environment file.

The tcadmin tool guides you through selecting parameters to be passed in the environment file.

---

## Setting up user exits

When you want TeamConnection to start a user exit, you must associate the user exit program with TeamConnection actions. We recommend you use the Family Administrator GUI to associate the actions with the program; however, if you prefer to do this manually, see 384. Instructions for using the GUI follow.



Go to “Setting up user exits” on page 384 to complete this task.

Follow these steps to associate user exit programs with TeamConnection actions. Before you do this task, we recommend that you stop the family server (refer to page “Stopping the servers” on page 95 for instructions).

1. Do one of the following to display the TeamConnection Family Administrator window:
  - From the TeamConnection Group folder on the desktop, double-click on the **Family Administrator** icon.
  - Type tcadmin from an OS/2 command prompt.
2. Display the family icon’s pop-up menu; then select **Settings**. The Settings notebook appears.

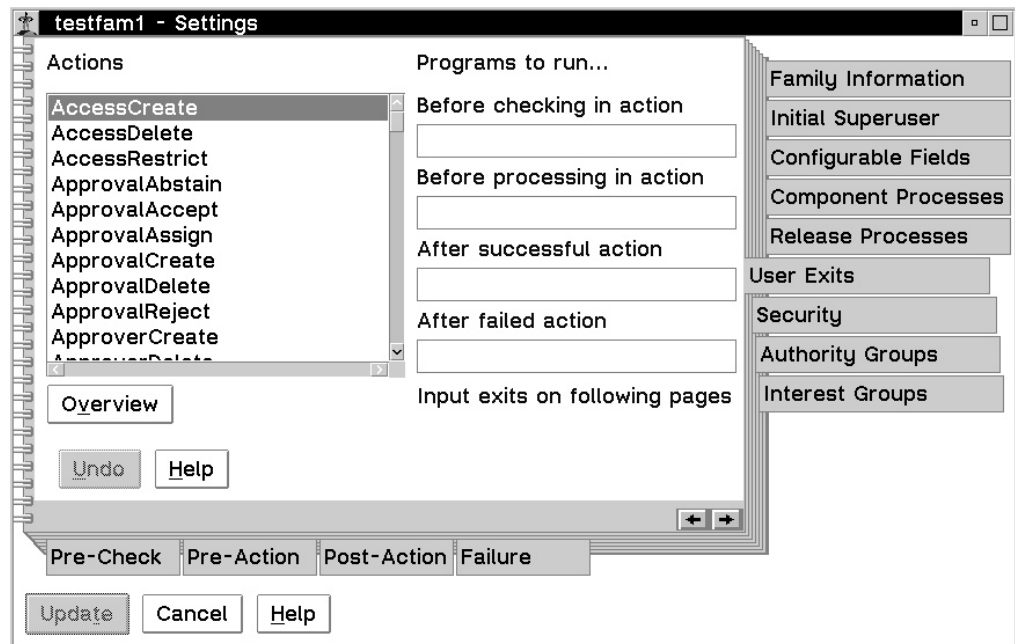


Figure 33. The User Exit settings page

3. Select **User Exits** to display the user exit page.
4. Highlight the action that you want to associate the user exit with, then select the tab at the bottom of the Settings page to specify when you want the user exit to start.

**Pre-Check (Exit ID 0)**  
before Action checking

**Pre-Action (Exit ID 1)**  
before Actions processing

**Post-Action (Exit ID 2)**  
after Action completes

**Failure (Exit ID 3)**  
if action or programs fail

Type the name of the user exit program to be executed as well as any user-defined parameters and comments (see “Configuring user exit parameters” on page 164 for more information on specifying parameters). Select the **Apply** push button to save the information for each user exit you specify.

Repeat this step for each user exit program that you want to start. Select **Overview** to see a complete listing of what user exit programs are defined and when they will start.

Select **Help** for more information about using the User Exits page.

5. When you finish making changes to your notebook pages, select **Update** (or **Create**) to save your changes and close the notebook.

The changes do not take effect until you start the family server.

---

## Configuring user exit parameters

You can configure the parameters to be passed to a user exit program for each user exit ID (0, 1, 2, or 3) as follows:

- Reorder the parameters
- Select which of the available parameters for an action are to be passed to the user exit and which are to be omitted
- Include configurable fields in the list of parameters to be passed to the user exit
- Include new fields that were previously not available

If you choose to configure the parameter list for a user exit ID, TeamConnection creates a new field in the userExit file that identifies the parameter list. Normally, a user exit definition in the userExit file appears as follows:

```
PartAdd 0 viewexit "1991 1992"
```

TeamConnection uses an ENV=( ) field to define a customized parameter list. If you want your user exit to be passed only the component and release parameters of the PartAdd action, for example, TeamConnection defines the user exit as follows:

```
PartAdd 0 viewexit "1991 1992" ENV=(component,release)
```

Any of the customizations listed at the beginning of this section will cause an ENV=( ) field to be generated for the user exit definition. The values for the customized parameters are stored in a temporary file so that they can be extracted into the user exit program.

To configure a parameter list for a user exit, follow these steps:

1. From the **User Exits** page of the Family Administrator Settings notebook, set up the user exit program for the action and the exit ID as described in “Setting up user exits” on page 162.
2. Using the notebook tabs at the bottom of the User Exits page, select the tab corresponding to the user exit ID for which you defined the user exit:

**Exit ID 0**  
**Pre-Check**

**Exit ID 1**  
**Pre-Action**

**Exit ID 2**  
**Post-Action**

**Exit ID 3**  
**Failure**

To configure a parameter list for the user exit associated with the Pre-Check exit ID of the PartAdd action, for example, select **PartAdd** from the list of actions on the User Exits page, then select the **Pre-Check** tab at the bottom of the page.

The following is an example of the Pre-Check notebook page for the PartAdd action.

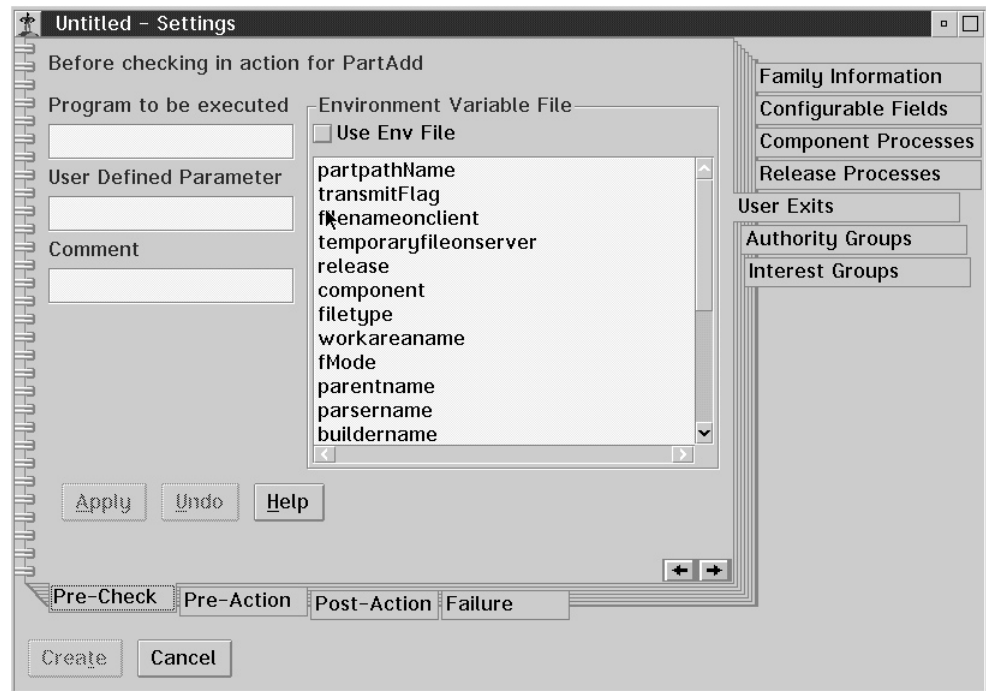


Figure 34. User exit parameters settings page

3. Set options for the Pre-Check exit ID for PartAdd as follows:

**Program to be executed**

Type the full path name of the user exit program.

**User Defined Parameter**

Type any parameters that you want to pass to the user exit program for the action and exit ID.

**Comment**

Type a comment to be included in the userExit file with the user exit definition.

**Environment Variable File**

To use a customized parameter list, first select the **Use env file** checkbox, then select the parameters you want to use from the list. If any configurable fields have been defined, they appear at the bottom of this list. The values for these parameters will be stored in a temporary file for extracting into the user exit program.

4. Select the **Apply** push button to save the options you have set.

The options that you set on this page are added to the userExit file. They take effect after you stop and restart the family server.

## Sample user exit programs

TeamConnection is shipped with the following sample user exit programs:

**samples/teamcenv.c**

Extracts the value of customized parameters from the temporary environment file so that they can be passed to a user exit program. You can incorporate this code into any user exit programs written in C.

**samples/viewexit.c, samples/viewexit.ksh, and samples/viewexit.cmd.**

Each of these samples shows the following information:

- Parameter 0: Executable name
- Parameter 1: User defined parameters
- Parameter 2: Environment variable file name (generated by TeamConnection and deleted automatically after the daemons are brought down)

Each sample also can output the contents of the environment variable file (viewexit.c has a simple function to dump the contents, while viewexit.cmd and viewexit.ksh have commented out lines that can call teamcenv).

The list of available user exits and their parameters can be viewed by a super user issuing the following command:

```
teamc report -userExitInfo
```

Adding the -long flag will also display the user exits currently configured and the environment variables to be written to the environment file.

---

## Part 4. Maintaining the TeamConnection server

<b>Chapter 16. Maintaining your TeamConnection environment</b>	169
Changing the age of defects and features	169
The age utility	169
The resetAge utility	170
Resolving TeamConnection errors	170
Using the error log	170
Using the audit log	170
Cleaning up the audit log	175
Using the trace facility	176
Maintaining the TeamConnection database	177
osbackup	178
oschangedbref (oschange)	180
UNIX	180
Windows and OS/2	180
Moving to the same directory	181
Moving to different directories	181
Moving to a different server	181
oscp	181
osrestore (osrestor)	182
ossevol	184
ossize	185
ossvrping (ossvrpin)	186
ossvrstat (ossvrsta)	186
osverifydb (osverify)	186
Using a raw file system (rawfs) database	187
Creating a rawfs database	187
Changing an existing file database to a rawfs database	188
 <b>Chapter 17. Migrating to TeamConnection version 2</b>	189
Migrating from TeamConnection version 1 to version 2	190
General guidelines	190
Preparing the source server for migration	191
Setting up the target server	193
Performing the migration	196
Migrating the current version of objects	196
Migrating previous versions	202
Executing migtc.lst	207
Preparing to administer your new database	210
Tables for authority, interest, configurable fields, and processes	210
User exit file	211
Configurable field definitions	211
Mail exits	211
Initialization file	211
REXX command files	211
Importing fine-grained data into your migrated database	211
Migrating from CMVC to TeamConnection version 2	212
General guidelines	213
Preparing the source server for migration	213
Converting SCCS keywords	215
Setting up the target server	218
Performing the migration	222
Migrating the current version of files from CMVC	223
Migrating previous versions	226

Executing migcmvc.lst . . . . .	227
Preparing to administer your new database . . . . .	229
Tables for authority, interest, configurable fields, and processes . . . . .	232
User exit file . . . . .	232
Configurable field definitions . . . . .	233
Mail exits. . . . .	233
Database schema file . . . . .	233
Using the migration tools . . . . .	233
Migration tool variables . . . . .	235
Sample migration files . . . . .	236
Migrate command . . . . .	237
Report command. . . . .	242
Set command . . . . .	242
Select command . . . . .	243
Update command . . . . .	244
Delete command . . . . .	244
Describe command . . . . .	244
Exec command . . . . .	245
! command . . . . .	245
Quit command. . . . .	246
# command . . . . .	246
<b>Chapter 18. Monitoring family use.</b> . . . .	247
Using the license monitor. . . . .	247
How the license monitor counts users . . . . .	248
Using the tclcmmon command . . . . .	248
Reporting highest uses . . . . .	249
Displaying a full use report . . . . .	250
Examples . . . . .	251
Using the server daemon monitor. . . . .	253
Using the monitor command . . . . .	253
Monitoring the activity of the server daemons . . . . .	254
Detecting time-consuming requests . . . . .	255
Monitoring server daemon problems. . . . .	255

This section contains information on maintaining your TeamConnection database, monitoring family use, and migrating your database from CMVC or TeamConnection version 1 to TeamConnection version 2.



---

## Chapter 16. Maintaining your TeamConnection environment

This chapter tells you how to do the following:

- Change the age of defects and features
- Take care of returned mail
- Resolve TeamConnection errors
- Maintain the TeamConnection database

---

### Changing the age of defects and features

TeamConnection provides two aging utilities: `age` and `resetAge`. Use these utilities to update the age value of defects and features while work is in progress. If you do not use these utilities, the age value for each defect and feature remains at zero.

Before you use the age utilities, make sure you have set the `TC_DBPATH` and `TC_FAMILY` environment variables in your `config.sys` file as follows:

- Set `TC_DBPATH` to the directory where the family database is. Make sure that you do not include a semicolon ( ; ) or backslash ( / ) at the end of this path.
- Set `TC_FAMILY` to the family name.

The following is an example for a family database named `testfam`:

```
SET TC_DBPATH=c:\teamc\testfam
SET TC_FAMILY=testfam
```

With these environment variables, the age utilities will change the defect and feature ages of the TeamConnection family database file `testfam.tcd` in the directory `c:\teamc\testfam`.

### The age utility

Use the age utility to increment the age value by 1 for each defect or feature that is in a specified state.

The `age.cmd` file is located in the directory where the TeamConnection server is installed. Initially, the file is set up to update the age of defects that are in one of the following states:

- open
- working
- design
- size
- review

You can edit the file to delete one or more of these states or to add any of the following states:

- canceled
- returned
- closed
- verify

Run the age utility from a server machine using the following command:

age

## The resetAge utility

Use the resetAge utility to reset the age of defects and features based on their state (open, working, design, size, and review), the date they were opened, and the selected aging increment.

Run the resetAge utility from a server machine using the following command.

```
resetage ageIncr
```

Where *ageIncr* is one of the following:

fullweek

Ages the defects and features according to a 7-day schedule.

workweek

Ages the defects and features according to a 5-day work week schedule.

---

## Resolving TeamConnection errors

The TeamConnection library includes the *Messages* book that lists messages that include recovery information. Use this book to help you resolve TeamConnection messages. If you require further help, refer to TeamConnection's error log or audit log. This section explains how to use these two logs, and also briefly explains the trace facility.

### Using the error log

Severe errors that are encountered by the family server are recorded in the syslog.log file. Use this file to help you better understand and resolve the error. This log usually provides more information than what is found in the initial message. There is only one syslog.log file, so if you have multiple families, error information for each family is recorded in the same file.

### Using the audit log

For each family, TeamConnection provides an audit log that contains an entry for every action performed since the family was created. In OS/2 and Windows environments, the audit.log file is located in the directory where your family database is installed (your TC\_DBPATH).



In AIX and HP-UX environments, the audit log is located in the directory structure /audit/log in your family database directory.

The audit log file contains information about both successful and unsuccessful transactions, making it useful for determining the source of a problem. It also includes an entry whenever an unauthorized attempt is made to access the TeamConnection server. This can help you audit your system's security.

The following information is recorded in the audit log for each transaction:

- For authorized transactions:
  - Process ID number of the family server
  - TeamConnection action
  - Whether the transaction was successful or not
  - Date of the transaction
  - Time the transaction started
  - Time the transaction ended
  - User ID of the person who requested the action
  - The name of the host system from which the user is accessing TeamConnection
  - Additional information for successful transactions, or error messages for unsuccessful transactions. See page 171 for the additional information about each TeamConnection action.
- For unauthorized transactions:
  - Process ID number of the family server
  - User ID of the person who requested the action
  - The name of the host system from which the user is accessing TeamConnection
  - Notification that the request was unauthorized
  - Date and time of the transaction request
  - Error message

The following is an example of information as it appears in the audit.log file:

```
31381,ReleaseCreate,FAILURE,1996/03/17,11:06:34,11:06:44,tcserv,tcserv,alexm.ral.ibm.com,
0010-427 A release already exists or previously existed with the name robot.
Specify a new name for this release.
31410,ReleaseCreate,SUCCESS,1996/03/17,11:07:05,11:07:12,tcserv,tcserv,alexm.ral.ibm.com,robot_control
31417,PartAdd,FAILURE,1996/03/17,11:07:32,11:07:35,tcserv,tcserv,alexm.ral.ibm.com,
0010-258 The requested action requires that you specify one or more
defects or features.
0010-263 File FILEH1.bin associated with release robot_control cannot be created.
31423,PartAdd,SUCCESS,1996/03/17,11:08:18,11:08:19,tcserv,tcserv,alexm.ral.ibm.com,FILEH1.bin,robot,1
31450,PartCheckOut,SUCCESS,1996/03/17,11:09:03,11:09:03,tcserv,tcserv,alexm.ral.ibm.com,FILEH1.bin,robot,1.1
31240,PartCheckIn,SUCCESS,1996/03/17,11:09:56,11:09:57,tcserv,tcserv,alexm.ral.ibm.com,FILEH1.bin,robot,1.2
31425,PartUndo,SUCCESS,1996/03/17,11:11:54,11:11:55,tcserv,tcserv,alexm.ral.ibm.com,FILEH1.bin,robot,delta,1.3
31436,ReleaseCreate,SUCCESS,1996/03/17,11:32:50,11:33:00,tcserv,tcserv,alexm.ral.ibm.com,robot_v2
31446,PartLink,FAILURE,1996/03/17,11:33:17,11:33:21,tcserv,tcserv,alexm.ral.ibm.com,
0010-052 Part FILEH1.bin associated with release robot_v2 was not found.
31449,PartLink,SUCCESS,1996/03/17,11:33:41,11:33:42,tcserv,tcserv,alexm.ral.ibm.com,FILEH1.bin,relH1,robot_v2,1.2
31249,PartCheckOut,SUCCESS,1996/03/17,11:35:08,11:35:08,tcserv,tcserv,alexm.ral.ibm.com,FILEH1.bin,robot_v2,1.3
31259,PartCheckIn,SUCCESS,1996/03/17,11:35:18,11:35:18,tcserv,tcserv,alexm.ral.ibm.com,FILEH1.bin,relI1,1.4
24942,Transaction from joe/tcserv@tcserv.ral.ibm.com was UNAUTHORIZED,03/18/95,09:43:11,
0010-100 User joe was not found.
```

*Figure 35. Sample of an audit log file*

The following table lists the additional information that is provided for each TeamConnection action.

<b>TeamConnection Action</b>	<b>Additional information</b>
<b>AccessCreate</b>	TeamConnection user ID, component name, authority group name
<b>AccessDelete</b>	TeamConnection user ID, component name
<b>AccessRestrict</b>	TeamConnection user ID, component name, authority group name
<b>ApprovalAbstain</b>	Release name, defect or feature name, approver's name
<b>ApprovalAccept</b>	Release name, defect or feature name, approver's name
<b>ApprovalAssign</b>	Release name, defect or feature name, new approver's name
<b>ApprovalCreate</b>	Defect or feature name, release name, approver's name
<b>ApprovalDelete</b>	Defect or feature name, release name, approver's name
<b>ApprovalReject</b>	Release name, defect or feature name, approver's name
<b>ApproverCreate</b>	TeamConnection user ID, release name
<b>ApproverDelete</b>	TeamConnection user ID, release name
<b>CompCreate</b>	New component name
<b>CompDelete</b>	Component name
<b>CompLink</b>	Component name, new parent component name
<b>CompModify</b>	Component name
<b>CompRecreate</b>	Component name
<b>CompUnlink</b>	Component name, parent component name
<b>CompView</b>	Component name
<b>CoreqCreate</b>	Release name, first defect or feature name, second defect or feature name
<b>CoreqDelete</b>	Release name, defect or feature name
<b>DefectAccept</b>	Defect name
<b>DefectAssign</b>	Defect name
<b>DefectCancel</b>	Defect name
<b>DefectClose</b>	**This action is not audited**
<b>DefectComment</b>	Defect name
<b>DefectDesign</b>	Defect name
<b>DefectModify</b>	Defect name
<b>DefectOpen</b>	Defect name
<b>DefectReopen</b>	Defect name
<b>DefectReturn</b>	Defect name
<b>DefectReview</b>	Defect name
<b>DefectSize</b>	Defect name
<b>DefectVerify</b>	Defect name
<b>DefectView</b>	Defect name
<b>DriverAssign</b>	Driver name, release name, new driver owner's TeamConnection user ID
<b>DriverCheck</b>	Driver name, release name

<b>TeamConnection Action</b>	<b>Additional information</b>
<b>DriverCommit</b>	Driver name, release name
<b>DriverComplete</b>	Driver name, release name
<b>DriverCreate</b>	Driver name, release name
<b>DriverDelete</b>	Driver name, release name
<b>DriverExtract</b>	Driver name, release name
<b>DriverModify</b>	Driver name, release name
<b>DriverView</b>	Driver name, release name
<b>EnvCreate</b>	Tester's TeamConnection user ID, release name, environment name
<b>EnvDelete</b>	Environment name, release name
<b>EnvModify</b>	Tester's TeamConnection user ID, release name, environment name
<b>FeatureAccept</b>	Feature name
<b>FeatureAssign</b>	Feature name
<b>FeatureCancel</b>	Feature name
<b>FeatureClose</b>	**This action is not audited**
<b>FeatureComment</b>	Feature name
<b>FeatureDesign</b>	Feature name
<b>FeatureModify</b>	Feature name
<b>FeatureOpen</b>	Feature name
<b>FeatureReopen</b>	Feature name
<b>FeatureReturn</b>	Feature name
<b>FeatureReview</b>	Feature name
<b>FeatureSize</b>	Feature name
<b>FeatureVerify</b>	Feature name
<b>FeatureView</b>	Feature name
<b>FixActive</b>	Defect or feature name, release name, component name
<b>FixAssign</b>	Defect or feature name, release name, component name
<b>FixComplete</b>	Defect or feature name, release name, component name
<b>FixCreate</b>	Defect or feature name, release name, component name
<b>FixDelete</b>	Defect or feature name, release name, component name
<b>HostCreate</b>	TeamConnection user ID, host name, user login on host
<b>HostDelete</b>	TeamConnection user ID, user login on host, host name
<b>MemberCreate</b>	Driver name, defect or feature name, release name
<b>MemberDelete</b>	Driver name, defect or feature name, release name
<b>NotifyCreate</b>	TeamConnection user ID, component name, interest group
<b>NotifyDelete</b>	TeamConnection user ID, component name
<b>PartAdd</b>	Path name, release name, SID
<b>PartCheckIn</b>	Path name, release name, SID

<b>TeamConnection Action</b>	<b>Additional information</b>
<b>PartCheckOut</b>	Path name, release name, SID
<b>PartDelete</b>	Path name, release name
<b>PartExtract</b>	Path name, release name, SID
<b>PartForceIn</b>	**Audited via PartCheckIn**
<b>PartForceOut</b>	**Audited via PartCheckOut**
<b>PartLink</b>	Path name, release name, new release name, SID
<b>PartLock</b>	Path name, release name, SID
<b>PartLockForce</b>	**Audited via PartLock**
<b>PartModify</b>	Path name, release name
<b>PartRecreate</b>	Path name, release name
<b>PartRecreaForce</b>	**Audited via PartRecreate**
<b>PartRename</b>	Path name, new path name, release name
<b>PartRenameForce</b>	**Audited via PartRename**
<b>PartResolve</b>	Base name, release name
<b>PartUndo</b>	Path name, release name, undo type, SID
<b>PartUndoForce</b>	**Audited via PartUndo**
<b>PartUnlock</b>	Path name, release name
<b>PartView</b>	Path name, release name
<b>ReleaseCreate</b>	New release name
<b>ReleaseDelete</b>	Release name, new release name
<b>ReleaseExtract</b>	Release name, new release name
<b>ReleaseLink</b>	Release name, new release name
<b>ReleaseModify</b>	Release name, new release name
<b>ReleaseRecreate</b>	Release name, new release name
<b>ReleaseView</b>	Release name, new release name
<b>Report</b>	**With -where flag: view name, criteria **With -help flag: help, none **With -testClient flag: test, none **With -testServer flag: test, none
<b>SizeAssign</b>	Defect or feature name, component name, release name
<b>SizeAccept</b>	Defect or feature name, component name, release name
<b>SizeCreate</b>	Defect or feature name, component name, release name
<b>SizeDelete</b>	Defect or feature name, component name, release name
<b>SizeReject</b>	Defect or feature name, component name, release name
<b>TestAbstain</b>	Defect name, release name, environment name, tester's TeamConnection user ID
<b>TestAccept</b>	Defect name, release name, environment name, tester's TeamConnection user ID
<b>TestAssign</b>	Defect name, environment name, new tester's TeamConnection user ID

<b>TeamConnection Action</b>	<b>Additional information</b>
<b>TestReject</b>	Defect name, release name, environment name, tester's TeamConnection user ID
<b>WorkareaAssign</b>	Defect or feature name, release name, new work area owner's TeamConnection user ID
<b>WorkareaCancel</b>	Defect or feature name, release name
<b>WorkareaCheck</b>	Defect or feature name, release name, driver name
<b>WorkareaCommit</b>	Defect or feature name, release name
<b>WorkareaComple</b>	Defect or feature name, release name
<b>WorkareaCreate</b>	Defect or feature name, release name
<b>WorkareaFix</b>	Defect or feature name, release name
<b>WorkareaIntegra</b>	Defect or feature name, release name
<b>WorkareaModify</b>	Defect or feature name, release name, target
<b>WorkareaTest</b>	Defect or feature name, release name
<b>WorkareaView</b>	Defect or feature name, release name
<b>UserCreate</b>	New user ID
<b>UserDelete</b>	User ID
<b>UserRecreate</b>	User ID
<b>UserModify</b>	User ID
<b>UserView</b>	**No additional information is audited**
<b>VerifyAbstain</b>	Defect or feature name, TeamConnection user ID
<b>VerifyAccept</b>	Defect or feature name, TeamConnection user ID
<b>VerifyAssign</b>	Defect or feature name, TeamConnection user ID of the new verification record owner
<b>VerifyReject</b>	Defect or feature name, TeamConnection user ID

## Cleaning up the audit log

TeamConnection continually appends information to the end of the audit log. To keep this file from growing too large, type the following from a command line in the directory containing your TeamConnection family. If you need to maintain the audit.log for more than one TeamConnection family, then type this command from the directory where each family is located. Before issuing this command, stop the family server (refer to page “Stopping the servers” on page 95).

```
tccleanu fileSize
```

Where *fileSize* is the size of the specified file in bytes. If you do not specify the size, the default is 256000.

TeamConnection creates a backup file called audit1.log. It places this file in the directory where the family server is located and from which you issue the tccleanu command. You can rename this file to any name you want for archive purposes. If you do not rename the file, TeamConnection keeps three backup logs in addition to the current log: audit2.log, audit3.log, and audit4.log. Each time you run the tccleanu program, TeamConnection moves the contents of each log file as follows:

1. audit3.log information is moved to audit4.log.
2. audit2.log information is moved to audit3.log.
3. audit1.log information is moved to audit2.log.
4. audit.log information is moved to audit1.log.

After this command is issued, the audit.log file is empty and ready to log new information.

## Using the trace facility

TeamConnection provides environment variables for trace. Modify the trace environment variables *only* when directed to do so by an IBM service representative.

The names of the TeamConnection trace environment variables, the purpose they serve, and the TeamConnection component that uses the environment variable are listed in the following table:

Environment variable	Purpose	Used by
BSERV_LOG	Indicates which parts are to be traced for the build agent or processor. You can set this when directed to do so by an IBM service representative. Otherwise it is set to null.	Build agent and build processor
BSERV_TRACEATTEMPTS	Specifies maximum number of failed trace attempts, for the build agent or processor, accepted before giving up. You should not need to change this.	Build agent and build processor
BSERV_TRACEDELAY	Specifies the amount of time, in seconds, that TeamConnection waits, when a trace attempt for the build agent or processor fails, before attempting another trace. The default is 1 second. You should not need to change this.	Build agent and build processor
BSERV_TRACEFILE	Specifies the trace file path and name for the build agent or processor.	Build agent and build processor
BSERV_TRACESIZE	Specifies the maximum size of the trace file for the build agent or processor, in bytes. If reached, wrapping occurs.	Build agent and build processor
TC_TMP	Specifies the directory where the family server stores temporary files. This environment variable is used only in Windows environments.	Family server



Environment variable	Purpose	Used by
TC_TRACE	Specifies the variable that lets the user designate which parts should be traced. You should modify this only when directed to do so by an IBM service representative. Otherwise it is set to null.	Client and family server
TC_TRACEATTEMPTS	Specifies the maximum number of failed trace attempts accepted before giving up. You should modify this only when directed to do so by an IBM service representative.	Client and family server
TC_TRACEDELAY	Specifies the amount of time in seconds that TeamConnection waits, when a trace attempt fails, before attempting another trace. The default is 1 second. You should modify this only when directed to do so by an IBM service representative.	Client and family server
TC_TRACEFILE	Specifies the output (part path and name) of the trace that the user designates using TC_TRACE.	Client and family server
TC_TRACESIZE	Specifies the maximum size of the trace file in bytes. If this size is reached, wrapping occurs. The default is one million bytes.	Client and family server

## Maintaining the TeamConnection database

This section describes several ObjectStore utilities that you can use to maintain and tune your TeamConnection database. It contains brief descriptions of a portion of the available utilities. For a complete list of ObjectStore utilities and instructions for using them, refer to the ObjectStore documents listed in “Bibliography” on page 491.

This section describes the following utilities. The utility names are truncated for OS/2. The truncated name is shown in parentheses.:

### **osbackup**

Provides online backup of TeamConnection databases

### **oschangedbref (oschange)**

Changes external database references for a database

**oscp** Creates a copy of a TeamConnection database

### **ossevol**

Performs schema evolution on a database

### **ossize**

Displays cross-database references in a database

### **osrestore (osrestor)**

Restores databases that were backed up using osbackup

**ossvrping (ossvrpin)**

Reports whether the family server is running on a specified host

**ossvrstat (ossvrsta)**

Displays statistics on all clients currently connected to the family server, as well as information about the family server that is not specific to a particular client

**osverifydb (osverify)**

Verifies that pointers within a database are valid

The osbackup and osrestore utilities provide protection from catastrophic data loss due to hardware failure. They can also be used to transport databases from one location to another.

## osbackup

Use the osbackup utility to back up your TeamConnection databases. This utility transparently backs up TeamConnection to a secondary storage device while it is running without affecting concurrency.

If your family uses one or more remote databases in addition to the family database, you must backup all databases simultaneously.

Use the following command to back up your databases:

```
osbackup [options] -f backup_image_file... pathname...
```

Where:

- Options

- b** *blocking factor*

Specifies the blocking factor for tape input and output. The blocking factor is in units of 512-byte blocks. The default on UNIX is 126 blocks. The maximum you can specify is 512 blocks. This parameter is ignored for regular files.

- i** *incremental\_record\_file*

Specifies the incremental record file, a file that contains information about which databases have been backed up, and when they were backed up. The osbackup utility uses this information to determine which segments within a database have been modified since the last backup at a lower level. The utility then backs up only modified segments. The incremental record file is comparable to the archive record file for osarchiv. Performing a backup at any level for which no previous information exists is equivalent to doing a level 0 backup for that database. This option overrides the default name of the file that records this information.

The follow are default names for the file:

- For UNIX, \$OS\_TMPDIR/back9pup\_record.UID, where UID is the effective user ID of the user performing the backup.
- For Windows and OS/2, %OS\_TMPDIR%\OSBACKUP.REC

- I** *import\_file*

Specifies the name of a file that contains a list of either file or rawfs database pathnames. The osbackup utility backs up the databases in this list. If you specify "-" as the import file name, osbackup reads from standard input. The list contains one pathname per line. Leading and

trailing white space is ignored. If you specify the `-l` option, you can also specify additional pathnames on the command line.

- l level** Specifies the level of the backup. Specify an integer from 0 to 9. Files that have been modified since the last backup at a lower level are copied to the backup image. For example, suppose you did a level 2 backup on Monday, followed by a level 4 backup on Tuesday. A subsequent level 3 dump on Wednesday would contain all files modified or added since the level 2 (Monday) backup. Backup is incremental at the segment level, meaning that a segment is only backed up if it has been modified since the last backup at a lower level. A level 0 backup (the default) backs up all segments in all specified databases.
- r** Instructs `osbackup` to descend into any rawfs directories specified on the command line, adding all rawfs databases found to the list of databases to be backed up. By default, only databases in the specified directory are backed up. When backing up file databases, specifying the `-r` option has no effect. You must explicitly specify each file database.
- s size** Sets the size of the volume being dumped to. The `osbackup` utility prompts you to insert a new tape or specify a new backup image file after it writes the amount of data specified by size. You can specify `k`, `m`, or `g` to indicate that size is in units of kilobytes, megabytes (the default), or gigabytes. For example, `-s 1024k`, `-s 1m`, and `-s 1` each specify a maximum backup image size of 1 megabyte. You can use this option with the `-f` option to perform a multivolume backup. This option is mainly for use when you are backing up to a tape device, since end-of-media cannot be reliably detected on some systems. On Solaris 2, the `-s` option is not required because the end of the tape is reliably signaled to the application without any loss of data. On other systems, if you do not specify `-s`, the `osbackup` utility terminates when it reaches the end of the tape.
- **-f backup\_image\_file** is the location of the backup image. You can specify a local file or a locally mounted file. You can specify a tape device that is directly accessible from the host on which you are running `osbackup`. You cannot specify a remote tape device. You can repeat the `-f` option with a new backup-image-file to create a multifile backup. When you do this, specify the `-s` option to indicate the size of each backup-image-file, as in the following example:

```
osbackup -s 1m -f back1 -f back2 -f back3 db1db2 db3
```

ObjectStore tries to back up the databases to the `back1`, `back2`, and `back3` files. The utility prompts for additional file names if 3 MB is not sufficient.

On UNIX systems, you can specify `-f -` (hyphen) to indicate stdout. This allows you to pipe `osbackup` output directly to the `osrestore` utility.

On Windows NT systems, you specify a tape device with this syntax:

```
\\.\Tape0
```

Tape0 is the standard Windows NT name for the first tape drive.

- **pathname** is the fully qualified path name of the database to be backed up; for example, `c:\teamc\bin\testfam.tcd`. You must include the file extension in the path name. Wildcards are not supported. You can specify more than one path name. Path names can be on different servers.

If you are using a raw file system database, specify the path name as follows:

```
<hostname>::/<family>.tcd
```

For <hostname> specify the host name of the server machine; for <family> specify the name of your TeamConnection family.

## oschangedbref (oschange)

You can use the `oschangedbref` utility to change the location of a database. This utility changes the external database references for a database. Before you run this utility, run `ossiz` to display the cross-database references in the database whose references you want to change. Carefully examine `ossiz` output to determine how that database defines relative pathnames. Use the information obtained from `ossiz` to specify the `fromPathname` and `toPathname` arguments to the `oschangedbref` command.

The `oschangedbref` command has the following syntax:

```
oschangedbref db {fromPathname | -n name1} {toPathname | -n name2}
```

Where:

- *db* is the database for which you want to change references.
- *fromPathname* is the currently referenced database. Specify an absolute path name that includes a server host prefix.
- **-n name1** specifies a relative path name for the currently referenced database. You must use this option for names beginning with a hyphen.
- *toPathname* is the database to be referenced. Specify a relative or an absolute path name according to how the original reference was defined.
- **-n name2** specifies a relative path name for the database to be referenced. You must use this option for names beginning with a hyphen.

The following are examples of using the `oschangedbref` utility.

### UNIX

```
ossiz loon:/dbs/db_0
```

```
[ ...]
```

```
External database pointers:
```

```
Relative name db_1, resolves to loon:/dbs/db_1
```

```
External references:
```

```
Relative name db_3, resolves to loon:/dbs/db_3
```

To change the reference to `db_3` to a reference to `db_7`, enter the following:

```
oschangedbref loon:/dbs/db_0 loon:/dbs/db_3 loon:/dbs/db7
```

For the `to` argument, you can use the relative pathname instead of the absolute name. The following is an equivalent command:

```
oschangedbref loon:/dbs/db_0 loon:/dbs/db_3 db7
```

### Windows and OS/2

```
ossiz me:h:\temp\t_1
```

```
[output omitted]
```

```
External database pointers:
```

```
Relative name t_3, resolves to me:h:\temp\t_3
```

```
Relative name t_0, resolves to me:h:\temp\t_0
```

```
Relative name t_2, resolves to me:h:\temp\t_2
```

```
External references:
```

```
Relative name t_3, resolves to me:h:\temp\t_3
```

To change pointers and references from t\_3 to t3.new, enter the following command:

```
oschange me:h:\temp\t_1 me:h:\temp\t_3 me:h:\temp\t_3.new
```

or

```
oschange me:h:\temp\t_1 me:h:\temp\t_3 t_3.new
```

## Moving to the same directory

/a/b/db1 contains a reference to /a/b/db2. If you move both db1 and db2 to a different directory, for example, /e/f/g, the reference is still valid because the result is

```
/e/f/g/db1  
/e/f/g/db2
```

Both db1 and db2 are still in the same relative pathname. If you move db1 to /e/f and db2 to /e/f/g, the result is

```
/e/f/db1  
/e/f/g/db2
```

In this case, they are no longer in the same relative path. You need to use the `oschangedbref` utility.

## Moving to different directories

/a/b/db1 contains a reference to /a/b/c/db2. In this case, db1 refers to c/db2. So if you move db1 to /e/f and move db2 to /e/f/c the result is

```
/e/f/db1  
/e/f/c/db2
```

The reference is still valid because db1 still refers to c/db2.

## Moving to a different server

/a/b/db1 on server green contains a reference to /a/b/db2 on server green. You want to move db1 to server red. If you move them both to the same directory on server red, the reference is still valid.

If you want to move only db1 to server red in directory /x/y, then you must use `oschangedbref` to change the reference from db1 to specify the full pathname, including server name, for db2.

When you store references, no server name is attached until you use `oschangedbref` to specify it.

## oscp

Use the `oscp` utility to create a copy of a TeamConnection database. The database you specify as the *source\_pathname* is copied to the database you specify as the *destination\_pathname*. If the destination database already exists, it is deleted before the copy is performed.

The syntax of `oscp` is:

```
oscp [-R] [-i] source_pathname destination_pathname
```

Where:

- -R instructs oscp to copy a directory recursively. This option is used for rawfs databases only. The top-level name of the destination path name must exist before you issue the oscp command.
- -i instructs oscp to prompt you to confirm that you want to overwrite databases or directories at existing path names. The prompt appears only if the target already exists.
- source\_pathname is the database to be copied.
- destination\_pathname is the pathname for the copy. The destination is either created or overwritten.

Wildcards in the path name are not supported, nor can you specify a directory.

If you are using a raw file system database, specify the path names as follows:

```
<hostname>::/<family>.tcd
```

For <hostname> specify the host name of the server machine; for <family> specify the name of your TeamConnection family.

Before performing the copy, the utility verifies that the database being copied is transaction-consistent and fully up-to-date. When the database is copied, the copy receives a new, unique ID.

**Note:** Copying a rawfs database to a file database can result in the loss of segment-level access control information. This can happen because file databases do not maintain this information. The oscp utility issues a warning after copying a database if the source database had segment-level protections that could not be copied. If the source database is not using segment-level access control, nothing is lost and a warning is not displayed.

## osrestore (osrestor)

The osrestore utility restores databases that have been backed up using the osbackup utility. TeamConnection cannot access databases that are being restored until the entire restore process completes.

If your family uses one or more remote databases in addition to the family database, you must restore all databases simultaneously. Specify the path name of each database on the restore command. You must restore to the same path from which you backed up the database.

To restore databases, always begin with a full level 0 backup image; osrestore prompts for any additional incremental backup images you might want to apply. Not all incremental backups necessarily need to be applied. To determine which incremental backups to apply, list the backup levels in chronological order, starting with the level 0 backup.

For instance, if you made a level 0 backup on Monday, a level 5 on Tuesday and Wednesday, a level 3 on Thursday, and level 4 on Friday, your list would look like this: 0, 5, 5, 3, 4. Scan the list from right to left and find the lowest incremental backup level greater than 0. In this example, that is the level 3 backup made on Thursday. Therefore, all increments made between the level 0 backup and this level 3 backup need not be applied. To restore databases to their current state as of the backup on Friday, you must apply the level 0 backup and the incremental backups made at level 3 and 4, in that order.

Use the following command to restore your TeamConnection databases:

```
osrestore [options] -f backup_file [pathname_translation]
```

Where:

- Options

- b** *blocking factor*

Specifies the blocking factor for tape input and output. The blocking factor is in units of 512-byte blocks. The default on UNIX is 126 blocks. The maximum you can specify is 512 blocks.

- n** Restores only the databases contained in the directory specified in the *pathname* argument. If you do not specify this option, all databases in the specified directory and its subdirectories are restored.

- t** Prints a listing of databases contained in this backup image.

- f** *backup\_file* is the file or tape device that contains a backup image from which to restore the databases. This parameter is required.

On UNIX, you can specify -f - (hyphen) to indicate stdin.

- pathname\_translation* specifies a pair of path names separated by a space. The first path name in the pair indicates the source of the database as recorded in the backup image. The second path name indicates the target; that is, the path name for the database after it is restored.

If you are using a raw file system database, specify the path names as follows:

```
<hostname>:./<family>.tcd
```

For <hostname> specify the host name of the server machine; for <family> specify the name of your TeamConnection family.

You can specify zero, one, or multiple path name translations. If you do not specify at least one path name translation, all databases in the backup image are restored in their original locations.

You must specify a pathname translation when you restore or recover data on an architecture that is different from the architecture on which you are restoring the data. For example, here is a Windows NT to UNIX pathname translation. The backup image being restored is /tmp/my.img. The interaction is on a UNIX system. You do not need to do anything special when you make the backup on the Windows NT system.

In the first interaction, the command line specifies the -t option, which instructs the osrestore utility to list the databases in the specified backup image. Nothing is actually restored. The only database in the backup image is mckinley:e:\r4tsd\_data\arch.0. This is a Windows NT database, and the example below shows that the osrestore utility on a UNIX system translates it to mckinley:e:/r4tsd\_data/arch.0. The utility automatically translates back slashes (\) to slashes (/).

```
% osrestore -f /tmp/my.img -t
```

```
Recovering from volume #1 (/tmp/my.img)...
```

```
mckinley:e:/r4tsd_data/arch.0
```

```
Closing volume #1 (/tmp/my.img).
```

```
%
```

In the second interaction, the command line specifies the pathname translation mckinley:e:/r4tsd\_data/ /recovery. This instructs the osrestore utility to copy all files



in the backup image in the mckinley:e:/r4tsd\_data/ directory to the /recovery directory on the local machine. In this example, this is only arch.0.

```
% osrestore -f /tmp/my.img mckinley:e:/r4tsd_data/ /recovery
```

```
Recovering from volume #1 (/tmp/my.img)...
Restoring 3175 sectors to database "vancouver:/recovery/arch.0"
Recovered to time Fri Mar 3 14:07:24 1995
Do you wish to restore from any additional incremental backups?
(yes/no):
```

```
no
```

```
Closing volume #1 (/tmp/my.img).
%
```

## ossevol

Use the ossevol utility to perform schema evolution on a database. This utility modifies a database and its schema so that it matches a revised application schema. Schema evolution is a task that you need to perform if you have tool builders who have added new classes or attributes to the TeamConnection database using the TeamConnection Toolbuilder's Development Kit (TBDK). Before you use this utility, you need to have completed the following steps according to the instructions in the *Toolbuilder's Development Guide*:

1. Completed the code for your tool
2. Rebuilt the server dll, fhcschem.dll and copied it to your *teamcInstallPath*\dll subdirectory
3. Rebuilt schema.adb and copied it to your *teamcInstallPath*\bin subdirectory
4. *teamcInstallPath*\dll subdirectory
5. Stopped the family server
6. Used osbackup to backup your database

After these steps are complete, you can issue the ossevolve command to evolve the database schema. After issuing the ossevolve command, you can restart the family server.

The ossevolve command has the following syntax:

```
ossevolve workdb schemadb evolvedb [keywordOptions]
```

### Where:

- **workdb** is the name of the working database in which the new classes or attributes were created.
- **schemadb** is the rebuilt version of schema.adb.
- **evolvedb** is the production database that is the target of the schema evolution operation.
- **keywordOptions** are the following:

#### **-task\_list**

The name of the file to which the task list will be written. Specify "-" for stdout.

#### **-classes\_to\_be\_removed**

List the names of classes to be removed from the database.



**-classes\_to\_be\_recycled**

List the names of classes to be recycled. By default the storage associated with all classes is recycled.

**-local\_references\_are\_db\_relative**

Specify **yes** or **no**. If you specify **yes** all local references are assumed to be relative to the database in which they were encountered. The default is **no**.

**-resolve\_ambiguous\_void\_pointers**

Specify **yes** or **no**. If you specify **yes** ambiguous void pointers are resolved to the outermost enclosing colocated object. The default is **no**.

**-upgrade\_vector\_headers**

Specify **yes** or **no**. The default is **no**.

**-drop\_obsolete\_indexes**

Specify **yes** or **no**. If you specify **yes** any obsolete indexes encountered during the course of schema evolution are dropped. The default is **no**.

**-workspace**

Specify the workspace for versioned databases.

**-explanation\_level**

Specify a number in the range of 1 to 4. This options is primarily an internal debugging aid.

**-recycle\_all\_classes**

Specify **yes** or **no**. The default is **yes**.

**Example:**

The following is an example ossevolve command:

```
ossevolve work.tcd schema.adb familydb.tcd -drop_obsolete_indexes yes
```

## ossiz

Use the ossize utility to display the cross-database references in a database whose references you want to change. Carefully examine ossize output to determine how that database defines relative pathnames. Use the information obtained from ossize to specify the fromPathname and toPathname arguments to the oschangedbref command.

The ossize command has the following syntax:

```
ossiz [options]pathname
```

Where:

- [options] are the following:

**-c** Print the type Contents for each segment.

**-C** Print the type Contents for the entire database.

**-ss** Sort type summaries by the Space of each type. (This is the default.)

**-sn** Sort type summaries by the Number of objects in each type.

**-st** Sort type summaries by the Type name.

**-0** Include the internal segment 0 in type summaries. Implies -c if neither -C nor -c is set.

- w x** Examine the database with the current workspace set to x.
- W** Print the names of all Workspaces in this database.
- a** Detailed printout of All internal database structure.
- n n** Print only information about data segment Number n.
- o** Print size and location of every Object.
- f** Print map of Free space in each segment.
- A** Print access control information for each segment.
- *pathname* is the path name of the database for which you want to display database information.

## ossvrping (ossvrpin)

Use the ossvrping utility to verify that the family server is running on the specified host. The syntax is:

```
ossvrping [-v] [hostname]
```

Where:

- -v provides more information when the family server cannot be contacted.
- *hostname* is the name of the host for which you want to test the connection. If not specified, this defaults to the host from which the command is issued.

## ossvrstat (ossvrsta)

Use the ossvrstat utility to display statistics about all clients that are currently connected to the family server on the specified host. Family server resource information is provided for each client, followed by each client's state, grouped by state. This command also displays the parameter settings and usage meters for the family server.

In the report, each client is identified by its host name. If the program name is known, it is listed along with the process ID on that host. If the program name is not known, it is so noted.

The command syntax for ossvrstat is:

```
ossvrstat hostname [-meters] [-clients] [-parameters] [-rusage]
```

Where:

- *hostname* is the name of the host of the server for which you want information.
- -meters displays performance meters for the specified server.
- -clients displays the state of each client connection to the specified server and shows which clients, if any, are contending for locks.
- -parameters displays server parameter values.
- -rusage displays server process information. This option is for UNIX systems only.

## osverifydb (osverify)

Use the osverifydb utility to verify that pointers within a database are valid. It verifies that:

- There are no transient pointers.
- Persistent pointers point to valid, not deleted, storage.
- The declared type for a pointer as determined from the schema matches the actual type of the pointed-to object.

The command syntax for `osverifydb` is:

```
osverifydb [options] pathname
```

Where:

- `-coll` runs integrity checks to ensure that the collections in the database are valid.
- `-ignore_references` suppresses verification of references.
- `-illegal_pointer_action` sets the illegal pointer to null or uses the reference value that is supplied in response to the query.
- `-o` prints out the name of every object in the database.
- `-v` specifies to print every pointer value.
- `-w workspace_name` verifies the pointers for versioned databases in the specified workspace.
- `-limit number` limits the number of error messages that are reported for any segment in the database to the number you specify.
- `pathname` is the path name of the database that you are verifying. Wildcards are not supported.

If you are using a raw file system database, specify the path name as follows:

```
<hostname>::/<family>.tcd
```

For `<hostname>` specify the host name of the server machine; for `<family>` specify the name of your TeamConnection family.

When `osverifydb` detects a pointer that is not valid, it indicates the location and the value of the pointer. When possible, it prints out a symbolic path to the bad pointer, starting with the outermost enclosing object.

---

## Using a raw file system (rawfs) database

Maintaining a raw file system (rawfs) database provides a way for you to distribute your ObjectStore database across disks. Use a rawfs database only when the database is close to reaching the OS/2 2-gigabyte file size limit. For information on controlling database growth, see “Controlling database growth” on page 113.

You can either change an existing file database to a rawfs database or you can create a new rawfs database for your family.

## Creating a rawfs database

Do the following to create the rawfs partitions:

1. Do one of the following from the machine where the TeamConnection database is installed:
  - Select the **ObjectStore Setup** icon from the TeamConnection group folder.
  - Type `ossetup` at a prompt.
2. Select the **Shutdown Services** pushbutton to close any ObjectStore applications.

3. Select the **Create Rawfs Partition** pushbutton from the Setup window.
4. From the Setup Server Partitions window, select the **New** pushbutton and then specify the fully qualified name of the partition file. Also indicate the initial size of the partition file and check **Expandable** if you want the file to grow beyond its initial size. Select the **Create Rawfs Partition** pushbutton to create the first file partition. Information about the created file partition is displayed in the box. Repeat this step to create additional file partitions.
5. Select **Done**.
6. From the Setup window, select **Reinitialize Server** and then select **OK** from the Server Initialization window and confirm your request.
7. Select **OK** from the Information window.
8. Select **Exit** to exit ObjectStore Setup.
9. Shut down the workstation and then restart it.

You must create a locator file, *family\_name.loc*, in order for TeamConnection to find your rawfs file partitions. Add the following line to this file:

```
hostname::/family.tcd
```

Where:

- *hostname* is the hostname of the server machine.
- *family* is the name of your TeamConnection family.

Store the file in the directory pointed to by your TC\_DBPATH environment variable. TeamConnection first looks for the existence of a .loc file in that directory, and if not found, looks for a .tcd file.

For example, assume that your family name is tcfam and the value of TC\_DBPATH is g:\teamc. TeamConnection looks for g:\teamc\tcfam.loc and, if the file exists, determines the location of the rawfs. If the .loc file does not exist, TeamConnection looks for the file g:\teamc\tcfam.tcd.

## Changing an existing file database to a rawfs database

If you currently use a file database, do the following to change your existing file database to a rawfs database:

1. Use the osbackup command to backup the database.
2. Follow the instructions on page 1 on page 187 to create the rawfs partition.
3. Use the osmkdir command to create subdirectories in the rawfs partition. For example:  
osmkdir host::/teamc; osmkdir host::/teamc/db
4. Use the oscp command to copy the file into the partition. For example, to change the existing file database, tcfam.tcd, to a rawfs database, type the following:  
oscp c:\teamc\tcfam.tcd host::/teamc/db/tcfam.tcd

---

## Chapter 17. Migrating to TeamConnection version 2

### Note to Version 3 customers

Documentation for Version 3 migration utilities is not yet available.

TeamConnection provides tools for migrating from CMVC to TeamConnection (called migcmvc) and from TeamConnection version 1 to version 2 (called migtc). This chapter explains how to use these tools. It contains the following sections:

- Migrating from TeamConnection version 1 to TeamConnection version 2
- Migrating from CMVC to TeamConnection
- Using the migration tools

If you are currently using TeamConnection version 1 or CMVC, and you want to begin using TeamConnection version 2, then you must migrate your database to TeamConnection version 2.

- For TeamConnection version 1 customers, because of schema evolution changes between the TeamConnection version 1 databases and version 2 databases, you cannot simply install version 2 over your version 1 installation. Instead, you need to install TeamConnection version 2 on a different server from your version 1 server and migrate your version 1 database to the version 2 server using the migtc tool.
- For CMVC customers, although the TeamConnection product was developed as the next generation of CMVC, the underlying database has changed to the ObjectStore object-oriented database from Object Design, Inc. and much of the underlying functionality has changed. For this reason, you need to migrate your CMVC database to TeamConnection version 2. To migrate your database from CMVC to TeamConnection, you need to install TeamConnection version 2 on a different server from your CMVC server and migrate your CMVC database to the version 2 server using the migcmvc tool.

Refer to “Using the migration tools” on page 233 to learn more about the migration commands before beginning the migration process.

The following list of terms used in this chapter will help you understand the migration process:

#### **original database**

The original TeamConnection version 1 or CMVC database before migration.

#### **migrated database**

The new TeamConnection version 2 database after migration.

#### **source server**

The original family server (TeamConnection version 1 or CMVC) from which you are migrating your database.

#### **target server**

The new TeamConnection version 2 family server to which you are migrating your database.

#### **new source server**

A relocated source server. This term is used to refer to a source server that has been moved from its original production machine.



It is recommended that you make a backup copy of your family database and file structure before you begin migration. After you have backed up your family database, restart it in maintenance mode so that no updates can be made to it from client machines while the migration is being performed.

---

## Migrating from TeamConnection version 1 to version 2

TeamConnection version 2.0 cannot be installed over any previous version of TeamConnection. Instead you need to migrate your TeamConnection version 1 database to a new TeamConnection version 2 server. Existing TeamConnection customers must migrate their data if they want to be able to use it on TeamConnection version 2. To migrate to TeamConnection version 2, you need two separate server machines:

- A TeamConnection version 1.1 (or TeamConnection version 1.0 and the latest fixpack) family server with your original database. This server must also have a superuser ID and a host list for the superuser so that the target server can access the source server.

You must have the latest TeamConnection version 1.x fixpack installed before you begin migration.

- A separate TeamConnection family server with TeamConnection version 2 client and server components installed, to which you will migrate your database. This server must be installed and configured as described in “Part 2. Installing the TeamConnection server” on page 15. No database (.tcd file) should exist in the TC\_DBPATH location of the target server.

You can migrate your TeamConnection version 1 database to any platform supported by TeamConnection version 2.

## General guidelines

The user ID of the person responsible for the migration must have superuser authority on the source server to guarantee access to all data for migration. This authority will be migrated from the source database to the target database. The superuser must have a host list entry to the target server.

It is recommended that the migration be done in stages with frequent backups of the database. You can use where clauses to capture and migrate a limited or expanded set of data. With frequent backups, you can reset the database to the steps prior to this change and then repeat the migration with the modification.

If your TeamConnection version 1 database contains fine-grained data, you will need to export it to .cdf files and then import the .cdf file into your version 2 database. The import and export actions are not performed by migtc but by issuing TeamConnection line commands instead. Migrating fine-grained data requires careful planning. The following is one way you can incorporate migration of fine-grained data into the migration process:

1. On your source server, export your fine-grained data to .cdf files using the `teamc release -export` and `teamc workarea -export` commands.  
See “Preparing the source server for migration” for instructions.
2. On your target server, migrate your database to TeamConnection version 2.  
See “Performing the migration” on page 196 for instructions.
3. Copy the .cdf files from your source server to your target server.
4. Import the .cdf files into your migrated database on the version 2 target server.  
See “Importing fine-grained data into your migrated database” on page 211 for instructions.

## Preparing the source server for migration

There are several steps you can take before you begin migrating to help the migration process execute more smoothly:

- All drivers and work areas should be integrated and all drivers committed before beginning migration. All test records and verify records should be completed.
- CollisionViews are not migrated. In the 1.0 database, if a part is checked out in two different work areas, you need to resolve all collision records before migrating the part. Refer to the *User's Guide* for instructions.
- Create a user ID with superuser access to your original and migrated databases and a host list for the user ID. This ID will be used to access the database from the target server.
- You may want to run queries like the following and save the output to a file. These commands capture the data that will be migrated. You can experiment with where clauses to limit the objects to be migrated.

```
teamc report -view workareaView >workare1.view
```

```
teamc report -view releaseView >releasel.view
```

```
teamc report -view compView >compon1.view
```

```
teamc report -view defectView >defect1.view
```

```
teamc report -view featureView >feature1.view
```

- If you have fine-grained data in your original database, you need to export it to .cdf files. If you do not have fine-grained data, you can skip this step.

If you have releases with at least one integrated workarea, issue the following command on your source server for each release to export the release to a .cdf file. Specify a user ID with superuser authority on the -become attribute.

```
teamc release -export releaseName
             -file releaseName.cdf -become superUserID
```

If you have unintegrated work areas, issue the following command on your source server for each unintegrated work area to export the work area to a .cdf file:

```
teamc workarea -export workareaName -release
               releaseName -file workareaName.cdf -become superUserID
```

Be sure to record the following information for each export action:

- The name of the .cdf file
- For work areas, the name of the release it belongs to.
- Stop the family server and use the xcopy command with the /s and /e parameters (or an equivalent command) to create a backup copy of your TC\_DBPATH directory structure and all files in it (including schema.adb, all .ld files, all .fmt and .tbl files). Make sure these files do not get copied to your target server.
- Restart ObjectStore.
- Restart the family server in maintenance mode (using the -m option) so that no updates can be made to the server while migration is in progress. Client users can view information, but not change it.

TeamConnection version 1 databases are not portable. It is recommended that you keep the TeamConnection version 1 server installed and operational on its original machine. If you do move the version 1 database to a different machine prior to migration, please do the following:

- Use xcopy to copy the files in the source server's TC\_DBPATH directory to the new source server. Use the exact same TC\_DBPATH. The drive and directory for the database must exactly match the original.
- On the new source server, on the first line of the TCP/IP hosts file, place the IP address of the server followed by the hostname and address of the source server, a space, and then the alias, as follows:

```
1.11.111.111 testfam.company.com testfam
```



If you plan to use your source server as your TeamConnection version 2 server, then you need a new IP address and hostname for this machine, since its original hostname has been transferred to the new source server.

## Setting up the target server

You run the migration tools from the TeamConnection client component of the target server, which communicates with your source server to migrate the database.

Install TeamConnection version 2 on your target server according to the instructions in “Part 2. Installing the TeamConnection server” on page 15:

- If TeamConnection version 1 is installed on the machine you intend to use as your target server, you must uninstall it before installing TeamConnection version 2 and then reboot. If you remove a TeamConnection version 1 installation, you may have to manually delete some files from the TeamConnection version 1 installation directory.
- Install the TeamConnection version 2 client and server components and perform the installation verification procedure to create the testfam database. Make sure you complete the installation process, including updating your TCP/IP hosts and services files with IP addresses for your target family.
- Set or modify the following environment variables in config.sys or .profile on the target server:

**SET TC\_FAMILY=*migratedDatabaseName***

Set this variable to the name of your migrated family database on the target server.

**SET TC\_FAMILY\_CLIENT=*originalDatabaseName@hostname@port***

Set this variable to the family name, host name, and port ID of your original database on the source server.

**SET TC\_USER=*superuserID***

Set this variable to a user ID with superuser authority.

**SET TC\_BECOME= *superuserID***

Set this variable to a user ID with superuser authority.

**SET TC\_DBPATH=*migratedDatabasePath***

Set this variable to the directory path you created for your migrated database on the target server.

The following are examples of these environment variables, using v1dbase as the original family name and v2dbase as the migrated family name:

```
SET TC_FAMILY=v2dbase
```

```
SET TC_FAMILY_CLIENT=v1dbase@v1server@1111
```

```
SET TC_USER=migrator
```

```
SET TC_BECOME=migrator
```

```
SET TC_DBPATH=d:\teamc\v2dbase
```

- Reboot the machine.
- Determine the directory or path where the target database will be located. This directory path will be your TC\_DBPATH. Do not create a family database in this location. Instead just create a subdirectory from your TeamConnection version 2 installation directory to contain the migrated database. You could, for example, use the directory path d:\teamc\v2dbase.
- Set up a directory structure from your new subdirectory for your migrated database similar to the testfam sample structure described in the installation verification procedure. The following directory structure and files must exist off the TC\_DBPATH. For example, if TC\_DBPATH=d:\teamc\v2dbase, then the directory structure should appear as follows. You can copy these files from the \testfam subdirectory created when you ran the installation verification procedure for your version 2 installation.

```
teamc
```

```
v2dbase
```

```
schema.adb
```

```
cfgfield
```

Defect.fmt

Defect.tbl

Feature.fmt

Feature.tbl

Part.fmt

PartView.fmt

User.fmt

config

userExit

security (new file; did not exist in TeamConnection version 1)

**Note:** The TeamConnection administrator is responsible for migrating the .ld files after migration. See “Tables for authority, interest, configurable fields, and processes” on page 210

- When migrating a single, committed version of each file from TeamConnection version 1, you must first do a release -extract, file -extract, or part -extract, to get the files you want to migrate out of the source database. You must then move the files to the TeamConnection version 2 server machine to migrate them into the target database. You can move the files by using a copy command, ftp, or any other available method.
- Start the ObjectStore server. See the following for instructions.

Platform	Page
AIX	26
HP-UX	38
Solaris	49
OS/2	63
Windows	75

## Performing the migration

Once you have prepared your source and target servers, you are ready to perform the migration. The TeamConnection migration utility, migtc, is a command-line utility that you run from the TeamConnection client and server installed on your target server. It uses the information you provided in “Setting up the target server” on page 193 to locate and access the source database.

TeamConnection provides a file called migtc.lst containing sample migration commands. This file is located in the /bin subdirectory of your TeamConnection version 2 installation path. It may be helpful for you to start with this command file, modify it to suit your needs, and then execute the migration using this command file.

### Migrating the current version of objects

The following are sample contents of migtc.lst. The commands in this file are written for the following migration functions:

- All objects are migrated, for example, all users, all parts, all defects, and so on.
- Only the current versions of parts (files) are migrated.

If you want to select specific objects to be migrated, you will need to modify this command file by adding where clauses to the commands. See “Using the migration tools” on page 233 for information on specific migration commands.

```
set batchsize 100
```

```
set decache 1
```

```
set maxerrors 100
```

```
migrate Users
```

```
migrate HostView
```

```
migrate Authority
```

```
migrate Interest
```

```
migrate Cfgcomproc
```

```
migrate Cfgrelproc
```

```
migrate Config
```

```
migrate CompView where 1=1 order by addDate
```

```
migrate BcompView where name='root'
```

```
migrate ReleaseView
```

```
migrate EnvView
```

```
migrate AccessView
```

```
migrate NotifyView
```

```
migrate DefectView
```

```
migrate FeatureView
```

```
migrate BuilderView
```

```
migrate ParserView
```

```
migrate DriverView where state in ('complete', 'commit')
```

```
migrate WorkAreaView where state in ('commit','complete')
```

```
set top x:\sourceCodeTreeStructureforReleaseName1
```

```
migrate PartView -release yourReleaseName1
```

```
set top x:\sourceCodeTreeStructureforReleaseName2
```

```
migrate PartView -release yourReleaseName2
```

```
migrate FixView
```

```
migrate ApproverView
```

```
migrate ApprovalView
```

```
migrate SizeView
```

```
migrate TestView
```

```
migrate DriverMemberView
```

```
migrate VerifyView
```



```
migrate    NoteView
```

```
migrate    CoreqView
```

```
quit
```

To use this command file for your migration, you need to make the following minimal modifications. For a complete list of migration commands, see “Using the migration tools” on page 233.

- It is wise to divide this command file into several smaller files, such as `migt1.lst`, `migt2.lst`, and so on, and to back up your database after executing each chunk. Chunking this command file will enable you to backup your database at regular intervals during the migration.
- The `set top` command is used to point to the source code tree structure of parts when you migrate only the current version of parts. If you want to migrate only the current version of parts, do the following:
  1. Create the source code tree structure on the target server or on a LAN drive that the target server has read/write access to.
  2. Change the `set top` variable in `migt1.lst` to point to the directory structure.
- The following commands migrate `TeamConnection` parts. Modify these commands to include the release names defined in your original database. If you have only one release, delete one of these commands. If you have more than two, create additional `migrate PartView` commands.

```
migrate    PartView -release yourReleaseName1
```

```
migrate    PartView -release yourReleaseName2
```

If your original database contains fine-grained data, you need to modify these commands as follows to exclude the fine-grained data from the migration. After

the database is migrated, you will import the fine-grained data that you exported from your original database in “Preparing the source server for migration” on page 191 .

```
migrate PartView -release yourReleaseName1 where partType='file'
```

```
migrate PartView -release yourReleaseName2 where partType='file'
```

## Migrating previous versions

It is not recommended that you migrate all of your past versions, as this can be very time consuming, and you will most likely not use all of this information. Instead, keep your version 1 database as an archive and retrieve previous versions of parts when you need them. If you want to migrate multiple versions, however, you must have the latest fixpak installed for TeamConnection version 1.x.

To migrate previous versions, you need to create two migration command files containing the following commands. Make the minimum required changes to these command files as described previously. The `set top` variable must remain null, as shown in this example.

```
set top
```

```
set maxErrors 100
```

```
set decache 1
```

```
set batchsize 100
```

```
migrate users
```

```
migrate hostView
```

```
migrate authority
```

```
migrate interest
```

```
migrate cfgcomproc
```

```
migrate cfgrelproc
```

```
migrate config
```

```
migrate compView where 1=1 order by addDate
```

```
migrate bcompview where name='root'
```

```
migrate releaseView
```

migrate envView

migrate accessView

migrate notifyView

migrate defectView

migrate featureView

migrate builderview

migrate parserview

migrate driverview

```
migrate workareaview
```

```
migrate changeview
```

Execute the commands in this command file and then quit and save your database. See “Executing migtc.lst” on page 207 for instructions. Then execute the second command file containing the following commands:

```
migrate versionview where 1=1 order by adddate
```

```
migrate partfullview -release oneOfYourReleaseNames
```

```
migrate partfullview -release anotherRelease
```

```
migrate partfullview -release anotherRelease
```

```
migrate partview -release oneOfYourReleaseNames
```

```
migrate partview -release anotherRelease
```

```
migrate partview -release anotherRelease
```

```
migrate workareaview
```

```
migrate fixView
```

```
migrate approvalView
```

```
migrate approverView
```

```
migrate PartsOutView
```

```
migrate SizeView
```

migrate TestView

migrate DriverMemberView

migrate VerifyView

migrate CommonParts

migrate NoteView

migrate CoreqView

WorkareaView is migrated twice because the branchpoint needs to be migrated following migration of the versionView data.

## Executing migtc.lst

To migrate your database follow these steps:

1. Make sure your source and target servers are set up properly and the source server is running in maintenance mode.
2. Start the ObjectStore server on the target server.
3. Set TC\_NOAUTH=YES.

4. From the target server, run TeamConnection report commands against the source server to make sure you are connected:

```
teamc report -view users -family
sourceFamilyName
```

5. Ensure that migtc.lst and migtc.exe are in the \bin subdirectory of your TeamConnection version 2 installation path. Modify migtc.lst to suit your needs.
6. From a command prompt on your target server, type the following command to start the migration tool:

```
migtc
```

The migtc command displays a header and the migtc command prompt. The header contains information about your source and target servers. The migtc command prompt consists of the family name and > symbol.

```
testfam>
```

You can capture output to a file by starting the migration tool using the following command:

```
migtc 1>mig.out 2>&1
```

7. At the migtc command prompt, enter the following command to execute the migration commands in migtc.lst:

```
exec migtc.lst
```

As you migrate the database, you may notice the following messages and behavior:

- When users are migrated you will see one fhccomp record migrated. This is the root component for the family.
  - When VersionView is migrated, you may see a message that releaseName:1 and releaseName are already loaded in the target database. This is not an error.
  - When BuilderView is migrated, you will see a message for null builders indicating that there is no source to extract.
  - Migrating NoteView combines the actions from the history section into a single note. The number of notes migrated is likely to be less than the number of records for NoteView in the source database.
  - If you are migrating all version of parts, there may be more part extract actions than there are records in the report from the source database. This occurs for the following reasons:
    - There are additional part extracts for each part that has been changed in a work area.
    - There are additional part extracts for each part that has been changed in a driver before the driver was refreshed from the release.
  - If you are migrating all versions and your database contains parts with no contents, you will see five attempts to extract the part. The part will be migrated to the target after five attempts and the part type will be empty.
8. If you have partitioned this command file into several smaller files, backup your database between execution of each command file using the following command, or an equivalent:



```
copy d:\migratedDatabasePath\  
dbname.tcd d:\migratedDatabasePath\dbname2.tcd
```

Substitute your TC\_DBPATH for *migratedDatabasePath* and your database name for *dbname*. Give the target file for the copy command a different name each time you backup the database during the migration process.

9. After migration is complete, exit the migtc command interface and start the ObjectStore server on the target server.
10. Start the migrated family server on the target server.
11. In version 2, users can only view data if they have been granted access or are:
  - A superuser
  - A component owner
  - A part owner

You can grant users access to each component with the appropriate authority using a command like the following:

```
teamc access -create -login  
userName -authority authorityGroup -component componentName
```

12. Run queries like the following to verify that you have migrated all necessary data from the source database. Compare the output from these queries to the output from the queries you ran from the source server in “Preparing the source server for migration” on page 191.

```
teamc report -view workareaView >workare2.view
```

```
teamc report -view releaseView >release2.view
```

```
teamc report -view compView >compon2.view
```

```
teamc report -view defectView >defect2.view
```

```
teamc report -view featureView >feature2.view
```

For more information on the migration commands, see “Using the migration tools” on page 233.

## Preparing to administer your new database

In some cases you cannot use your existing TeamConnection version 1 configurable field or user exit files with your version 2 database. The following sections provide guidelines for migrating your existing files to your version 2 database or using the version 2 files.

### Tables for authority, interest, configurable fields, and processes

The `authorit.ld` and `interest.ld` files have been modified since the version 1 release. Generate new `.ld` files from the database by redirecting the raw output of a report. The following commands show how to create these files:

```
teamc report -raw -view config > config.ld
```

```
teamc report -raw -view authority > authorit.ld
```

```
teamc report -raw -view interest > interest.ld
```

```
teamc report -raw -view cfgrelproc > relproc.ld
```

```
teamc report -raw -view cfgcomproc > comproc.ld
```

Copy the `.ld` files to your `TC_DBPATH` where your migrated database exists. You should only use these `.ld` files if you want to add, delete, or change any of the

values. See “Creating or modifying authority groups” on page 129 (for instructions using the Family Administrator GUI) or “Creating or modifying authority groups” on page 371 (for instruction using family administrator line commands).

## User exit file

User exits from version 1 cannot be use with version 2. Use the userExit file shipped with TeamConnection version 2. Move the userExit file to the \config subdirectory of your TC\_DBPATH directory.

Any user exits written for version 1 will have to be rewritten since the infrastructure from version 1 to version 2 has changed. Use the sample viewexit.cmd located in the samples subdirectory of the TeamConnection installation path to see the new order of user exit parameters. You need to concern yourself mainly with any parameters being searched and returned. the viewexit.cmd sample will help you determine the new order of parameters.

## Configurable field definitions

If you are converting from TeamConnection version 1, and you have not made any changes to configurable fields in TeamConnection version 1, you should be able to use your version 1 .tbl files with version 2. If you have made changes, then you need to use the chfield command to update the tables. See “Creating and modifying configurable fields” on page 378 for instructions.

## Mail exits

If you are migrating from TeamConnection version 1, you can use your version 1 mailexit file unless you are also migrating to another platform. If you are migrating to another platform, use the TeamConnection version 2 mailexit file for your target platform.

## Initialization file

If you want to preserve queries you've written in your task list, you can use the teamc10.ini file. Change any part types of "file" to "TCPart."

```
copy teamc10.ini teamc20.ini
```

## REXX command files

If you have written any REXX command files for version 1, you will need to change any part types of "file" to "TCPart" before you use them with version 2.

# Importing fine-grained data into your migrated database

Once your database and all related files have been migrated to your target server, you can import your fine-grained data into it. The fine-grained data is contained in a set of .cdf files that you created in “Preparing the source server for migration” on page 191 . To import fine-grained data into your migrated database, follow these steps:

1. Copy the .cdf files from the source server to the target server.
2. Create one work area for each *releaseName.cdf* file created when preparing for the migration. To create a work area issue the following command:

```
teamc workarea -create -name workAreaName -release  
          releaseName -family familyName
```

Make sure you specify the release as defined in your original database. Refer to the notes you made in "Preparing the source server for migration" on page 191.

3. To import your releases, issue the following command for each *releaseName.cdf* file:

```
teamc workarea -import workareaName -release  
releaseName
```

```
-file releaseName.cdf -become superUserID
```

4. Integrate each work area you created for this process.
5. Create one work area for each *workAreaName.cdf* file created by issuing a teamc workarea -export command. To create a work area issue the following command:

```
teamc workarea -create -name workAreaName  
-release releaseName -family familyName
```

Make sure you specify the release as defined in your original database. Refer to the notes you made in "Preparing the source server for migration" on page 191.

6. To import your work areas, issue the following command for each *workAreaName.cdf* file:

```
teamc workarea -import workAreaName -release  
releaseName
```

```
-file workareaName.cdf -become superUserID
```

For -release *releaseName* make sure you specify the release to which the work area belonged in your original database. Refer to the notes you made in "Preparing the source server for migration" on page 191.

7. Integrate each work area you created for this process.

---

## Migrating from CMVC to TeamConnection version 2

Existing CMVC customers must migrate their data if they want to be able to use it on TeamConnection version 2. To migrate to TeamConnection version 2, you need two separate server machines:

- A CMVC family server with your original database. This server must also have a superuser ID and a host list for the superuser so that the target server can access the source server.
- A separate TeamConnection family server with TeamConnection version 2 installed, to which you will migrate your database. This server must be installed and configured as described in "Part 2. Installing the TeamConnection server" on page 15. No database (.tcd file) should exist in the TC\_DBPATH location of the target server. This server must also contain a CMVC 2.3 client with a superuser ID for accessing the source database.

Several technical reports containing further information about migrating from CMVC to TeamConnection are available:

**TR 29.2254**

Migration from CMVC 2.3 to TeamConnection 2.0

**TR 29.2232**

How to do migration tasks with CMVC

**TR 29.2253**

Comparison between CMVC 2.3 and TeamConnection version 2

These technical reports are available on the installation CD.

## General guidelines

The user ID of the person responsible for the migration must have superuser authority in the source system to guarantee access to all data for migration. This authority will be migrated from the source database to the target database.

It is recommended that the migration be done in stages with frequent backups of the database. You can use where clauses to capture and migrate a limited or expanded set of data. With frequent backups, you can reset the database to the steps prior to this change and then repeat the migration with the modification.

## Preparing the source server for migration

There are several steps you can take before you begin migrating to help the migration process execute more smoothly:

- Complete all levels and tracks before beginning migration to ensure accurate results.
- Tracks (in CMVC) in the integrate state are migrated to the fix state because there is no way of knowing which files have been checked out in relation to the track when moving to TeamConnection version 2. Fix records in the active state are ignored during migration. Ensure that tracks (in CMVC) are in the complete state before beginning the migration. For any tracks not in the complete state, the owner will have to manually checkout the files, replace them with the appropriate files containing the modifications, check the files into the work area, and integrate the workarea.
- Files from CMVC in the checkout or locked state are not migrated to the target database because the migrate tool does not know the work area or defect that the locked file is associated with. Before migrating, checkin or unlock all CMVC files.
- Level members in CMVC are migrated to a driver in the target database. Because only committed and complete levels are migrated to TeamConnection version 2, then level members for levels in the integrate state are not migrated. Ensure that all levels in the integrate state are committed or complete before migrating.
- Create a user ID with superuser access to your original database and a host list for the user ID. This ID will be used to access the database from the target server.
- Convert SCCS keywords in text files to be compatible with TeamConnection keyword support. TeamConnection provides several scripts that aid this process. These scripts can be found in the samples subdirectory.

- FileExtract2.migcmvc changes the keywords and generates a list of text files for all releases to be migrated. This script is added to the file -extract step of the migration process as a user exit to the file -extract command to be executed during migcmvc with change history (full) migration.
- keywordConversion.migcmvc is a script to be executed after performing a release extract to prepare for migrating files with keywords.

See “Converting SCCS keywords” on page 215 for complete instructions for converting keywords.

- You may want to run queries like the following and save the output to a file. These commands capture the data that will be migrated. You can experiment with where clauses to limit the objects to be migrated.

```
cmvc report -view trackView >track1.view
```

```
cmvc report -view releaseView >release1.view
```

```
cmvc report -view compView >compon1.view
```

```
cmvc report -view defectView >defect1.view
```

```
cmvc report -view featureView >feature1.view
```

- Restart the family server in maintenance mode (using the -m option) so that no updates can be made to the server while migration is in progress. Client users can view information, but not change it.

The following table shows how certain objects in a CMVC database are migrated into TeamConnection. These tables indicate which states can be migrated and how the states of objects might change from CMVC to TeamConnection.

*Table 22. Migrating CMVC objects to TeamConnection*

<b>Object</b>	<b>State in CMVC</b>	<b>State in TeamConnection</b>
Tracks	approve	approve
	cancel	cancel
	fix	fix
	integrate	fix
	commit	commit

Table 22. Migrating CMVC objects to TeamConnection (continued)

Object	State in CMVC	State in TeamConnection
	complete	complete
Fix	active	(not migrated)
Levels	working	not migrated
	integrate	not migrated
	commit	commit
	complete	complete

Defects in the working state prior to migration will be in the working state following migration. Any file changes that were checked into the database prior to migration but did not have completed fix records are not migrated. Following migration, the owner of the defect must create a work area, checkout the necessary files, make changes, check the changes in, and integrate the work area.

## Converting SCCS keywords

Keywords that have been inserted into files can be expanded during file extracts so that you can determine their version after the files have been delivered. This operation is accomplished by requesting the "Expand keywords" option during a file extract. Expanding keywords is essential in large systems where updates do not always replace all files in your product. CMVC and TeamConnection use different formats for keywords, so during a database migration, the CMVC keywords need to be converted to TeamConnection format. TeamConnection provides the following script that helps you convert CMVC keywords to TeamConnection format:

- keywordConversion.migcmvc prepares files with keywords for migration. You use this script only if you extract files before migrating.
- FileExtract2.migcmvc changes the keywords. This script is added to the file -extract step of the migration process as a user exit to the file -extract command to be executed during migcmvc with change history (full) migration.

These scripts use the following tools:

- sed to make changes to the files
- grep to make sure files need to be changed
- CMVC to find all files that have a file type other than "binary" (e.g. text, long, longSp, special) and have not been deleted

Do the following before you run these scripts, set the CMVC\_FAMILY environment variable to your CMVC family name and set the CMVC\_TOP environment variable to the directory path to which you are extracting.

There are two ways to use these scripts:

- If you extract levels or releases before you begin the migration process, use these scripts as follows:
  1. Extract the files to directory structures using the level or release extract command. Do not use the file extract command.
  2. Run keywordConversion.migcmvc to prepare files before the migration. Start this script as follows:
 

```
keywordConversion.migcmvc CMVC_FAMILY CMVC_TOP
```

Only .c and .ksh text files will be converted. You will have to modify the script to convert other file types. Binary files will not be converted.

3. Run migcmvc.

- If you extract files as part of a change history migration, use these scripts as follows. When using this scenario to convert keywords, the you must start the CMVC family with at least 2 daemons. This is due to the fact that the user exit will issue report commands that otherwise will deadlock the family.
  1. Add user exit FileExtract2.migcmvc to file -extract user exit ID 2. (See “Chapter 15. Providing user exits” on page 155 for instructions on setting up a user exit.
  2. Run migcmvc.

The file /tmp/checkreps.releaseName is a structured file containing the release and path name for each text file in all releases to be migrated. The script appends to this file for each release. Be sure to delete /tmp/checkreps.releaseName each time you want to generate a new list of files in it.

Keywords are converted to TeamConnection format as follows:

- The concept of current date/extract date is not relevant to TeamConnection. The last updated date is used during extract. As a result, the extract dates are commented out and the update dates are used:

```
%D% -> /* Use $ChkD, get date NA */
```

```
%H% -> /* Use $ChkD, get date NA */
```

```
%T% -> /* Use $ChkD, get date NA */
```

```
%E% -> $ChkD;
```

```
%G% -> $ChkD;
```



`%U% -> $ChkD;`

- Some of the keywords do not have a direct translations in SCCS (they were derived from PVCS):

`$Own;`

- All references to internal information exposed by SCCS are not converted to TeamConnection external values. As a result several values convert to a single `FileName:`

`%F% -> $FN;`

`%M% -> $FN;`

`%P% -> $FN;`

- Versioning information has been consolidated. As a result, the following keyword changes occur:

`%I% -> $Ver;`

`%W% -> $Ver;`

`%R% -> /* Use $Ver */`

`%B% -> /* Use $Ver */`

`%S% -> /* Use $Ver */`

- Ideally, the sed script should locate the last keyword and insert the end-replace keyword \$EKW after it. However, that is a personal preference and left for users to modify as they see fit.

If the conversion scripts detect keywords not supported, they will generate comments stating this fact. The following is a brief description of some of the SCCS keywords:

**\$Own;**

The owner of the component that manages the part.

**\$KW;** Begin keyword expansion after this keyword.

**\$EKW;**

End keyword expansion until the next \$KW; keyword. %Z% SCCS keyword converts to \$KW; in TeamConnection.

**%A%** Not applicable in TeamConnection

**%Y%** Not applicable.

**%C%** Not available in TeamConnection

## Setting up the target server

You run the migration tools from the target server, which communicates with your source server to migrate the database. Although the version 2 installation program cannot detect if a CMVC server is installed, you need to make sure that you install the version 2 server on a different machine from your CMVC server.

Follow these guidelines to ensure that your target server is set up properly for migration.

Install TeamConnection version 2 on your target server according to the instructions in “Part 2. Installing the TeamConnection server” on page 15:

- Install TeamConnection version 2 and perform the installation verification procedure to create the testfam database. Make sure you complete the installation process, including updating your TCP/IP hosts and services files with IP addresses for your target family.
- Install the CMVC 2.3 client on the target server. Make sure your config.sys or .profile contain the following settings:
  - Ensure that CMVC is in the PATH statement before TeamConnection, for example,

```
SET PATH=C:\CMVC\EXE;E:\teamcpath;
```

- Make sure CMVC.CAT is in the NLSPATH statement before TEAMCV20.CAT, for example,

```
SET NLSPATH=c:\cmvc23\exe\nls\%N;c:\teamcpath\nls\msg\enu\%N;
```

Several executable programs that are common to both the CMVC and target client are used in the migration. The PATH environment must find the appropriate version of these programs:

**File.exe**

To migrate files

**Report.exe**

To migrate all views

**Defect.exe**

To migrate notes

**Feature.exe**

To migrate notes

The CMVC file path should precede the TeamConnection version 2 file path.

- Determine the directory or path where the target database will be located. This directory path will be your TC\_DBPATH. Do not create a family database in this location. Instead just create a subdirectory from your TeamConnection version 2 installation directory to contain the migrated database. You could, for example, use the directory path d:\teamc\v2dbase.
- Set the following environment variables in config.sys or .profile on the target server:

**SET TC\_FAMILY=***migratedDatabaseName@hostname@port*

Set this variable to the name of your migrated family database on the target server.

**SET CMVC\_FAMILY=***originalDatabaseName@hostname@port*

Set this variable to the family name, host name, and port ID of your original database on the source server.

**SET TC\_USER=***superuserID*

Set this variable to a user ID with superuser authority.

**SET CMVC\_USER=***superuserID*

Set this variable to a user ID with superuser authority.

**SET CMVC\_BECOME=** *superuserID*

Set this variable to a user ID with superuser authority.

**SET TC\_BECOME=** *superuserID*

Set this variable to a user ID with superuser authority.

**SET TC\_DBPATH=***migratedDatabasePath*

Set this variable to the directory path you created for your migrated database on the target server.

The following are examples of these environment variables, using cmvcdbase as the original family name and v2dbase as the migrated family name:

```
SET TC_FAMILY=v2dbase@v2server@1111
```

```
SET CMVC_FAMILY=v1dbase@v1server@9999
```

```
SET CMVC_USER=migrator
```

```
SET CMVC_BECOME=migrator
```

```
SET TC_USER=migrator
```

```
SET TC_BECOME=migrator
```

```
SET TC_DBPATH=d:\teamc\v2dbase
```

- Reboot the machine.
- Set up a directory structure from your new subdirectory for your migrated database similar to the testfam sample structure described in the installation verification procedure. The following directory structure and files must exist off the TC\_DBPATH. For example, if TC\_DBPATH=d:\teamc\v2dbase, then the directory structure should appear as follows. You can copy these files from the \testfam subdirectory created when you ran the installation verification procedure for your version 2 installation.

```
teamc
```

```
v2dbase
```

schema.adb

cfgfield

Defect.fmt

Defect.tbl

Feature.fmt

Feature.tbl

Part.fmt

PartView.fmt

User.fmt

config

userExit

security (new file; did not exist in CMVC)

**Note:** The TeamConnection administrator is responsible for migrating the .ld files after migration. See “Tables for authority, interest, configurable fields, and processes” on page 210

- When migrating a single, committed version of each file from CMVC, you must first do a release -extract, file -extract, or part -extract, to get the files you want to migrate out of the source database. You must then move the files to the TeamConnection version 2 server machine to migrate them into the target database. You can move the files by using a copy command, ftp, or any other available method.
- Start the ObjectStore server. See the following for instructions.

Platform	Page
AIX	26
HP-UX	38
OS/2	63
Windows	75

## Performing the migration

Once you have prepared your source database and set up your source and target servers, you are ready to perform the migration. The TeamConnection migration utility, migcmvc, is a command-line utility that you run from your target server. It uses the information you provided in “Setting up the target server” on page 218 to locate the source database.

TeamConnection provides a file called migcmvc.lst containing sample migration commands. This file is located in the /bin subdirectory of your TeamConnection version 2 installation path. It may be helpful for you to start with this command file, modify it to suit your needs, and then execute the migration using this command file.

## Migrating the current version of files from CMVC

The following are sample contents of migcmvc.lst as shipped. The commands in this file are written for the following migration functions:

- Only the current versions of parts (files) are migrated.
- All objects are migrated, for example, all users, all parts, all defects, and so on.

If you want to select specific objects to be migrated, you will need to modify this command file by adding where clauses to the commands. See “Using the migration tools” on page 233 for information on specific migration commands.

```
set batchsize 100
```

```
set decache 1
```

```
migrate Users
```

```
migrate HostView
```

```
migrate Authority
```

```
migrate Interest
```

```
migrate Cfgcomproc
```

```
migrate Cfgrelproc
```

```
migrate Config
```

```
migrate CompView where 1=1 order by addDate
```

```
migrate bCompView where name='root'
```

```
migrate ReleaseView
```

```
migrate EnvView
```

```
migrate AccessView
```

```
migrate NotifyView
```



```
migrate DefectView
```

```
migrate FeatureView
```

```
migrate LevelView where 1=1 order by commitDate
```

```
migrate TrackView
```

```
migrate FixView
```

```
migrate ApproverView
```

```
migrate ApprovalView
```

```
migrate LevelMemberView
```

```
set top x:\location of files in release if migration only the current version
```

```
migrate fileview where releasename='9604' and dropdate is nullmigrate SizeView
```

```
migrate NoteView
```

The migration tool supports migration of committed versions of CMVC files directly into the release. The bulk contents are obtained from a local drive. To migrate CMVC files, you need to do the following:

- If you have a very large database, it may be wise to divide this command file into several smaller files, such as migcmvc1.lst, migcmvc2.lst, and so on. Chunking this command file will enable you to backup your database in intervals during the migration.
- Set the CMVC\_TOP environment variable or use the migrate command to set the top directory to the directory containing the CMVC files.
  1. Create the source code tree structure on the target server or on a LAN drive that the target server has read/write access to.
  2. Change the set top variable in migcmvc.lst to point to the directory structure.
- Use the set batchSize command to specify the number of files to process in one transaction.
- Use the set decache command to specify how often (after how many transactions) to decache the ObjectStore cache manager.
- Use the migrate FileView command to specify the release to be migrated from the CMVC database to TeamConnection. Files are read from the local drive. They are not extracted from CMVC when migrating a single version. Always include **dropdate is null** with your migrate command to ensure that it does not attempt to select files that have been deleted. You do not need to migrate or create a work area before migrating CMVC files. Modify this command to include the release names defined in your original database. If you have more than one release, create additional migrate fileview commands.

To use this command file for your migration, you need to make the following minimal modifications. For a complete list of migration commands, see “Using the migration tools” on page 233

## Migrating previous versions

It is not recommended that you migrate all of your past versions, as this can be very time consuming, and you will most likely not use all of this information. Instead, keep your CMVC database as an archive and retrieve previous versions of parts when you need them.

To migrate previous versions, the set top variable must remain null, as shown in this example.

```
migrate    LevelView
```

```
migrate    TrackView
```

```
migrate    ChangeView
```

```
set top
```

```
migrate    FileView -release releaseName
```

## Executing migcmvc.lst

To migrate your database follow these steps:

1. Make sure your source and target servers are set up properly, the source server is running in maintenance mode, and the client on the target server is running.
2. From the CMVC client installed in the source server, run CMVC report commands to make sure you are connected:

```
report -view users -family  
      sourceFamilyName
```

```
report -view hostView -family sourceFamilyName
```

3. Ensure migcmvc.lst is in the \bin subdirectory of your TeamConnection version 2 installation path. Modify migcmvc.lst to suit your needs.
4. From a command prompt in the directory where you copied migcmvc.lst, type the following command to start the migration tool:

```
migcmvc
```

The migcmvc command displays a header and the migcmvc command prompt. The header contains information about your source and target servers. The migcmvc prompt consists of the family name and > symbol.

```
testfam>
```

You can capture output to a file by starting the migration tool using the following command:

```
migcmvc 1>mig.out 2>&1
```

5. At the migcmvc command prompt, enter the following command to execute the migration commands in migcmvc.lst:

```
exec migcmvc.lst
```

As you migrate the database, you may notice the following messages and behavior:

- When users are migrated you will see one fhccomp record migrated. This is the root component for the family.
  - Migrating NoteView combines the actions from the history section into a single note. The number of notes migrated is likely to be less than the number of records for NoteView in the source database.
6. If you have partitioned this command file into several smaller files, backup your database between execution of each command file using the following command, or an equivalent:

```
copy d:\migratedDatabasePath\  
dbname.tcd d:\migratedDatabasePath\dbname2.tcd
```

Substitute your TC\_DBPATH for *migratedDatabasePath* and your database name for *dbname*. Give the target file for the copy command a different name each time you backup the database during the migration process.

7. After migration is complete, exit the migcmvc command interface and start the ObjectStore server on the target server.
8. Start the migrated family server on the target server.

**Note:** You are using the shipped version of configurable field files. If you have configurable fields, they may not display until the administrator has modified these files.

9. Grant users access to the data in the migrated database. In version 2, users cannot access data as they did in CMVC if they are not:
  - A superuser
  - A component owner
  - A part owner

You will need to grant users access to each component with the appropriate authority using a command like the following:

```
teamc access -create -login  
userName -authority authorityGroup -component componentName
```

10. Run queries like the following to verify that the correct number of objects were migrated. Compare the output from these queries to the output from the queries you ran from the source server in “Preparing the source server for migration” on page 191.

```
teamc report -view workareaView >workare2.view
```

```
teamc report -view releaseView >release2.view
```

```
teamc report -view compView >compon2.view
```

```
teamc report -view defectView >defect2.view
```

```
teamc report -view featureView >feature2.view
```

For more information on the migration commands, see “Using the migration tools” on page 233.

## Preparing to administer your new database

After migrating your database, you need to set up a directory structure for your family similar to the testfam sample structure described in the installation verification procedure. The following directory structure and files must exist off the TC\_DBPATH. Type **SET TC\_DBPATH** to get the name of the database directory. For example, if TC\_DBPATH=d:\teamc\testfam, then the directory structure should appear as follows:

teamc

testfam

schema.adb

*database.tcd*

(administrator is responsible for the .ld files)

cfgfield

Defect.fmt

Defect.tbl

Feature.fmt

Feature.tbl

Part.fmt

PartView.fmt

User.fmt

config

userExit

security (new file; did not exist in CMVC)

In most cases you cannot use your existing CMVC files with your version 2 database. The following sections provide guidelines for migrating your existing files to your version 2 database or using the version 2 files.

### **Tables for authority, interest, configurable fields, and processes**

You cannot use the configurable files as they existed in CMVC. You need to create and modify these files for TeamConnection. You can create the .ld files from the database by redirecting the raw output of a report. The following commands show how to create these files:

```
teamc report -raw -view config > config.ld
```

```
teamc report -raw -view authority > authorit.ld
```

```
teamc report -raw -view interest > interest.ld
```

```
teamc report -raw -view cfgrelproc > relproc.ld
```

```
teamc report -raw -view cfgcomproc > comproc.ld
```

Copy the .ld files to your TC\_DBPATH where your family database exists. You should only use these .ld files if you want to add, delete, or change any of the values. See “Creating or modifying authority groups” on page 129 (for instructions using the Family Administrator GUI) or “Creating or modifying authority groups” on page 371 (for instruction using family administrator line commands).

### **User exit file**

If you are converting from CMVC, use the TeamConnection version 2 config\userExit file and add to it any modifications that you made for your CMVC installation. See “Chapter 15. Providing user exits” on page 155 for instructions on working with user exits.



## Configurable field definitions

The CMVC configurable field files (defectConfigFormat, featureConfigFormat, fileConfigFormat) are replaced by a collection of .fmt files (defect.fmt, feature.fmt, part.fmt, and user.fmt). An additional configurable field file, called partView.fmt, is available in TeamConnection version 2. If you are migrating from CMVC, use the TeamConnection version 2 .fmt files and add to them any modifications that you made for your CMVC installation. See “Creating and modifying configurable fields” on page 378 for instructions on working with these configurable field files.

The configurable field tables from CMVC (ConfigTable files) can be used with TeamConnection version 2. Refer to the *CMVC Administrator's Guide* for the name and location of the files. The corresponding files should then be copied and renamed then placed in the cfgfield directory: defect.tbl, feature.tbl, user.tbl, part.tbl.

## Mail exits

Use the TeamConnection version 2 mailexit file for your target platform.

## Database schema file

Use the TeamConnection version 2 schema.adb file.

---

## Using the migration tools

The CMVC and TeamConnection migration tools provide a command line environment from which you can issue the migration commands. You start the migration tool using the version 2 client, which communicates with the version 1 server to migrate data to the version 2 server. To start the migration tools, type one of the following from your operating system command line:

### **migcmvc**

To start the migration tool for CMVC-to-TeamConnection

**migt** To start the migration tool for TeamConnection-to-TeamConnection

These commands present a prompt consisting of the family name and > symbol. Type the migration commands at this prompt. To see a list of the migration commands, type ? at this prompt, as in the following example:

```
testfam> ?
```

Valid commands are:

```
migrate view[,view...] [[where] where-clause]
```

```
migrate bcompview where name='top-component or root'
```

```
report viewName [query]
```

```
set argument [value]
```

```
select field[,field] from view [where query]
```

```
update view set attribute=value where expression
```

```
delete [from] view [where] expression
```

`describe view`

`exec file`

`!system-command`

`dump counters`

`quit`

For more information, enter the command followed by a "?"

To see additional help information for a single command, type the command followed by ?.

## Migration tool variables

Certain environment variables for the migration tool can be set to change how the tool operates. These variables use default values, but you can change them to suit your needs. The following is a list of the default values. For information on these variables and instructions for setting them, see "Set command" on page 242

`batchSize` set to 1000.

`maxErrors set to 0.`

`decache set to 0.`

`top set to "D:\".`

`reportStyle set to "stanza".`

`loadMessage set to "none".`

## Sample migration files

Sample files, called `migtclst` and `migcmvc.lst`, are provided with the install package. You can edit the commands in these files and use them to perform a migration. Use the `exec` command to execute the commands in these files:

```
exec migtclst
```

```
exec migcmvc.lst
```

To add comments to migration command files, use a # symbol. Any line preceded by a # symbol is ignored by the migration tool.

## Migrate command

The migration command executes a report command against the source client and parses this data while creating the objects in the TeamConnection target database. You can choose to migrate all or part of the source data. You select the data to be migrated by specifying where clauses on the migrate command to generate reports on the data to be migrated. Use caution when writing these where clauses, because there is potential for error if all objects, such as components or releases, are not migrated but files relating to these objects are migrated. It is your responsibility to generate the correct set of data for migration.

To migrate data from CMVC or TeamConnection to TeamConnection version 2, type the following command at the migration command prompt:

```
migrate view[,view...] [[where] whereClause]
```

Where:

- *view* is one of the views shown in the following list.
- *whereClause* is a whereClause constructed using column (or field) names as shown in the *Commands Reference*. Refer to the *Commands Reference* for more information on writing queries on TeamConnection views.

The following are some examples of the migrate command:

- The following command migrates all users:  

```
migrate users
```
- The following command migrates all components whose dropDate is not null.

```
migrate compview where dropDate is not null
```

- The following command migrates all defects in open and working state:

```
migrate defectview where state in ('open', 'working')
```

You can migrate the following views using the migrate command:

### **AccessView**

Access list data. Used for migration from TeamConnection version 1 and CMVC.

### **ApprovalView**

Approval records. Used for migration from TeamConnection version 1 and CMVC.

### **ApproverView**

Approver records. Used for migration from TeamConnection version 1 and CMVC.

## Authority

Authority list. Used for migration from TeamConnection version 1 and CMVC. During the process of migrating Authority, the actions defined in the authority table are changed to the TeamConnection action name as follows:

- Track actions are changed to work area actions
- Level actions are changed to driver actions
- File actions are changed to part actions

New actions that exist in TeamConnection but not in the source database are added and paired with an appropriate group. After migration, you can issue a TeamConnection report command to show the actions and authority groups that exist. These can be compared to a similar report from your source database.

Following migration, if the authority table needs modifications, you can produce an `authorit.ld` file as described in “Appendix A. Family administration commands” on page 369.

## bCompView

Component members which make up the component hierarchy. Used for migration from TeamConnection version 1 and CMVC. When migrating components (CompView) followed by component members (bCompView), it is required that the top level component be specified on the migrate command. For example, if the default top level component exists in the source database and is named `root` then, you need to migrate the component members using the following command:

```
migrate bcompview where name='root'
```

To see which component members will be migrated for the top-level component, issue the following report command:

```
report -view bcompview -where "name='yourRootComponent'" -raw
```

where *yourRootComponent* is the name of the root component on the source database. If you are not sure of the name of the top level component, the following query on the source client will provide it:

```
report -view compview -where "id not in (select
```

```
childid from compmemberview where parentid<>childid) and id in (select
```

```
parentid from compmemberview where parentid<>childid)" -raw
```

If the name of the top component is not root and root does not exist as a component, then you should modify the component name to make it root.

**BuilderView**

Builders are migrated and the corresponding file is extracted and created in the target database. Used for migration from TeamConnection version 1.

**Cfgcomproc**

Configurable component processes. Used for migration from TeamConnection version 1 and CMVC.

**Cfgrelproc**

Configurable release processes. Used for migration from TeamConnection version 1 and CMVC. When migrating configurable processes using Cfgrelproc, the name of the process will remain exactly as it was in the source database. For example, track\_level process will be named track\_level, but the process now includes "driver" instead of "level." If you wish to change the process name, it is recommended that you make a duplicate of the current process and name the new one track\_driver. To do this, first generate a relproc.ld file as described in "Appendix A. Family administration commands" on page 369. In this file, change track\_level to track\_driver and then execute the following command to reload the configurable release processes table:

```
fhclproc relproc.ld databaseName r
```

where *databaseName* is the path name of the target database. See section "Reloading the configurable process tables" on page 374 for more information.

**ChangeView**

All versions of files. Used for migration from CMVC.

**CommonParts**

Common parts Used for migration from TeamConnection version 1.

**CompView**

Components. Used for migration from TeamConnection version 1 and CMVC. See further instructions on bCompView.

**Config**

Configurable field values. Used for migration from TeamConnection version 1 and CMVC.

**CoreqView**

Corequisites. Used for migration from TeamConnection version 1.

**DefectView**

Defects. Used for migration from TeamConnection version 1 and CMVC.

**DriverMemberView**

Driver members. Used for migration from TeamConnection version 1.

**DriverView**

Drivers. Used for migration from TeamConnection version 1.

**EnvView**

Environments. Used for migration from TeamConnection version 1 and CMVC.

**FeatureView**

Features. Used for migration from TeamConnection version 1 and CMVC.

**FileView**

Files. Used for migration from CMVC.

**FixView**

Fix records. Used for migration from TeamConnection version 1 and CMVC.

**HostView**

Host lists. Used for migration from TeamConnection version 1 and CMVC.

**Interest**

Interest table. Used for migration from TeamConnection version 1 and CMVC. During the process of migrating the Interest, the actions defined in the interest table are changed to TeamConnection action names as follows:

- Track actions are changed to work area actions
- Level actions are changed to driver actions
- File actions are changed to part actions

New actions that exist in TeamConnection but not in the source database are added and paired with an appropriate group. After migration, you can issue a TeamConnection report to show the actions that exist. These can be compared to a similar report from your Source database.

Following migration, if the interest table needs modifications, you can produce an interest.Id file as described in "Appendix A. Family administration commands" on page 369.

**LevelMemberView**

Level members. Used for migration from CMVC.

**LevelView**

Levels. Used for migration from CMVC.

**NoteView**

Used for migration from TeamConnection version 1 and CMVC. The migrate NoteView command iterates over the collection of defects and features that have been migrated and issues a **defect -long -view defectName** or **feature -long -view FeatureName** command against the source database. It then searches for the word "history:" and takes all the following lines of data and creates a single note in the TeamConnection database. The user ID for the note is the originator of the defect or feature and the addDate of the defect or feature.

If you do not want to migrate all the notes, then migrate defects and features in stages. For example, if you want only notes for defects and features in the 'open' and 'working' state, then migrate defects and features "where state in ('open','working')"

and then migrate NoteView. Then proceed to migrated defects and features "where state not in ('open','working')".

**NotifyView**

Notification records. Used for migration from TeamConnection version 1 and CMVC.



**ParserView**

Parsers. Used for migration from TeamConnection version 1. When migrate of ParserView is complete, the parser files must be moved manually from the source server to the new TeamConnection server and located in the appropriate directory.

**PartFullView**

All versions of parts. Used for migration from TeamConnection version 1.

**PartView**

Parts. Used for migration from TeamConnection version 1. Issue one migrate PartView command for each release in your original database. Specify the release whose parts you want to migrate by including a -release attribute with the command as follows:

```
migrate PartView -release yourReleaseName
```

If you have fine-grained parts defined in your database, then you need to exclude them from the migration by specifying this command as follows:

```
migrate PartView -release yourReleaseName where partType='file'
```

See “Preparing the source server for migration” on page 191 for more information on preparing a database containing fine-grained data for migration.

**ReleaseView**

Releases. Used for migration from TeamConnection version 1 and CMVC.

**SizeView**

Sizing records. Used for migration from TeamConnection version 1 and CMVC.

**TestView**

Test records. Used for migration from TeamConnection version 1 and CMVC.

**TrackView**

Tracks from CMVC migrate to work areas. Used for migration from CMVC.

**Users** Used for migration from TeamConnection version 1 and CMVC. Do not use a where clause when migrating users. The **migrate users** command is a required step because it migrates the superuser authority and creates the special user named InheritedAccess. After you issue this command a component called root is created after all users are migrated. The root component is the top-level component and is used as the parent of the first component. The top level component on your target database must be

named root, the default name that is created when the database is first initialized. This should not be modified.

Since users are the owners, originators, and creators of most objects in the database, they must also exist in the target database in order for the related objects to be migrated. For this reason it is recommended that all users be migrated and that you avoid using the where clause.

#### **VerifyView**

Verification records. Used for migration from TeamConnection version 1 and CMVC.

#### **WorkAreaView**

Work areas. Used for migration from TeamConnection version 1.

## **Report command**

Data in the target database can be displayed with the report command. The format of the data depends on the setting of the reportStyle option (see “Set command”). If the view contains configurable fields, the fields will be displayed with titles like configField1, configField2, and so on, because the title of the fields is not known to the migration tool. The set reportStyle command determines the format of the report output.

To generate a report on the target database, type the following command at the migration command prompt:

```
report view [query]
```

Where:

- *view* is one of the views shown in “Migrate command” on page 237.
- *query* is a query constructed using column (or field) names as shown in the *Commands Reference*. Refer to the *Commands Reference* for more information on writing queries on TeamConnection views.

The following are some examples of the report command:

- The following command reports information on all users:

```
report users
```

- The following command reports information on all components whose dropDate is not null.

```
report compview where dropDate is not null
```

## **Set command**

To set options for the migrate and report commands, use the set command as follows:

```
set option value
```

Where:

- *option* is one of the options listed below.
- *value* is a valid value for the option.

You need to issue one set command for each option that you want to change. The following list shows the options you can set and the possible values for each. To display current values for each option, type **set ?**.

**batchSize**

This option specifies the number of objects to migrate in one transaction. The default is 1000.

**maxErrors**

The number of errors found before a migrate view is halted. For example, if migrating defects and the originator's user login is not found, then this counts as 1 error. The default is 0.

**reportStyle**

Specifies how reports generated by the report command are to be formatted. The default is stanza.

- **terse**
- **stanza**

**loadMessage**

Specifies how messages generated during a migration are to be formatted. The default is none.

- **terse**
- **stanza**
- **none**

**decache**

Specifies how often (after how many transactions) to decache the ObjectStore cache manager. The default is 0.

**deadlockRetry**

Specifies how many times to retry a transaction if a deadlock occurs. This variable can be set to any value.

**top** Specifies the location of the source code, if needed.

## Select command

Use the select command to display objects in a view. You use this command with the view names and field names described in the appendix of the *Commands Reference*

```
select field[,field] from
```

```
view [where query]
```

Where:

- *field* is a column name from the view.
- *view* is the name of the view from which you want to display objects.
- *query* is a query constructed using column (or field) names as shown in the *Commands Reference*. Refer to the *Commands Reference* for more information on writing queries on TeamConnection views.

The following example displays the login IDs for all users in the database

```
select * from users
```

## Update command

Use the update command to change the value of a field in the database. You use this command with the view names and field names described in the appendix of the *Commands Reference*

```
update view set
```

```
attribute=value where expression
```

Where:

- *view* is the name of the view whose values you want to update.
- *attribute* is the field name of the value you want to update.
- *value* is the new value.
- *expression* selects the rows from the table (or view) for which you want to update values.

The following example updates the superUser field for all users whose login ID is cpersche.

```
update users set superUser='yes' where login='cpersche'
```

## Delete command

Use the delete command to delete an object from the database. You use this command with the view names and field names described in the appendix of the *Commands Reference*

```
delete [from] view [where]  
expression
```

Where:

- *view* is the name of the view whose objects you want to delete.
- *expression* selects the rows from the table (or view) that you want to delete.

The following example deletes all users from the database whose login ID is minnie.

```
delete from users login in ('minnie')
```

## Describe command

Use the describe command to display the column names for a view.

```
describe view
```

Where:

*view* is the name of the view whose relationships you want to display. The following command shows the column names for the users view. These column names can then be used in select and where clauses.

```
describe users
```

## Exec command

Use the exec command to issue migration commands from a file.

```
exec  
  filepath
```

Where:

*filepath* is the full path name of the command file to execute. If you do not specify a full path name, the migration tool looks for the file in the current directory.

The following example executes migration commands defined in a file called migtc.lst.

```
exec migtc.lst
```

The following example executes migration commands defined in a file called d:\teamc\bin\migtc.lst.

```
exec d:\teamc\bin\migtc.lst
```

The file migtc.lst is a sample of the migration views that can be used to migrate your database.

## ! command

Use the ! command to issue operating system commands from the migration command prompt.

```
!systemCommand
```

Where: *systemCommand* is an operating system command.

The following example issues a dir command from the migration command prompt:

```
!dir
```

## Quit command

Use the quit command to exit the migration command prompt and return to the operating system command prompt.

```
quit
```

## # command

Use the # command to add a comment to a migration command file.

---

## Chapter 18. Monitoring family use

TeamConnection provides monitoring tools that enable you to keep track of how family servers are being used:

- A license monitor command (tclicmon) for gathering information from the audit log concerning the number of users who have contacted a TeamConnection family in a given time interval.
- A daemon monitor command (monitor) for monitoring the activity of the TeamConnection server daemons in real time.

---

### Using the license monitor

Use the TeamConnection license monitor to obtain a snapshot of the number of users who have contacted a TeamConnection family in a given time interval. The license monitor obtains family use information from the audit log. By default, only the audit.log for the current family is processed, but you can request information for another family on the same server.

**Note:** The license monitor needs to use an audit log file that is not currently in use by a family server. If the audit log is in use, stop the family server before running the license monitor.

The license monitor is a command that allows TeamConnection family administrators to monitor compliance with the terms of your license agreement by showing the number of concurrent uses of TeamConnection for a given time period. It is assumed that the family administrators know how many licenses the company obtained for TeamConnection.

The number of concurrent users is defined as the number of users who have contacted a TeamConnection family in a given amount of time. The default is 15 minutes. If, for example, you have 30 licenses and a total pool of 100 users, then up to 30 users can work with a TeamConnection family for any given period of 15 minutes.

The license monitor command does not enforce the limit of the number of licenses. Even if the number of actual users exceeds the number of licenses for TeamConnection, no attempt is made to limit access to a TeamConnection family. It is the responsibility of the family administrator to monitor the license usage and if the number of concurrent users exceeds the number of licenses for TeamConnection, then the family administrator should contact IBM to obtain more licenses. The number of licenses and the highest actual number of concurrent users should match.

The license monitor command is invoked from the directory of the family you want to monitor. It uses the contents of the audit.log to determine how many users (defined by each unique combination of user ID, login ID, and host name) have contacted the TeamConnection family in a given date and time interval, according to periods of a given duration (also called histograms). If, for example, use is to be monitored for two hours from 08:00 to 10:00, then the license monitor checks the audit log for users each 15 minutes (the default): four times per hour or eight times in the two-hour interval.

The following is an example of how family use might be reported for this two-hour period:

```
From 08:00:01 to 08:15:00, actual users: 1
From 08:15:01 to 08:30:00, actual users: 5
From 08:30:01 to 08:45:00, actual users: 5
From 08:45:01 to 09:00:00, actual users: 10
From 09:00:01 to 09:15:00, actual users: 8
From 09:15:01 to 09:30:00, actual users: 5
From 09:30:01 to 09:45:00, actual users: 3
From 09:45:01 to 10:00:00, actual users: 4
```

The highest amount of users in a given period is ten.

## How the license monitor counts users

A user is any unique combination of user ID, login ID, and host name. If, a user accesses the family using two user IDs from a single host name, for example, then that is counted as two separate users.

If more than one period in the interval being monitored has the same highest number of users, then only the first occurrence of that number is reported. If, for example, you monitor family use for three hours and the highest number of uses reaches twenty for two separate fifteen-minute periods, only the first occurrence is reported.

Because most TeamConnection transactions have a short duration, only the starting time for the transaction is considered by the license monitor command. When a long transaction starts in one time period and ends in another time period, the license monitor counts that use only once. It ignores the user's transaction for the second time period. A transaction in the audit log is processed only for those entries with a status of SUCCESS.

## Using the `tclicmon` command

You can issue the license monitor command any time, but it is recommended that you issue it at least once a day, especially before or after the daily backup of the family.

When you issue the command, you specify values for the dates and times that mark the interval you want to monitor. Use the following format for dates and times in the license monitor command:

```
yyyy/mm/dd,hh:mm:ss
```

The comma between the date and the time is required. The default value for the begin date and time is today at 00:00:01, and the default value for the end date and time is today at the current time.

The license monitor command has three action flags:

### **`tclicmon -highest`**

Displays only a summary of the report of concurrent users. The main element of the report is the time period that had the maximum use.

### **`tclicmon -report`**

Displays a full report of use for all time periods between the `-begin` date and the `-end` date for the duration specified in the `-timePeriod` attribute. You can request the report in several different formats.



### **tclicmon -help**

Displays a summary of the command, showing some examples and the defaults. To see help for the syntax, enter the command without any arguments.

**Note:** The order of the arguments for the tclicmon command needs to follow the sequence described in the syntax. For example, if you want to use a long report format with a begin date, then the order is:

```
-report -long -begin
```

If you change the order of the command arguments as follows:

```
-report -begin -long
```

the command will not be executed and a usage message will appear.

## **Reporting highest uses**

To display the highest use for an interval, issue the following command from the directory containing the family you want to monitor:

```
tclicmon -highest  
[-begin yyyy/mm/dd,hh:mm:ss]  
[-end   yyyy/mm/dd,hh:mm:ss]  
[-timePeriod minutes]  
[-input fileName]
```

Where:

- -begin *yyyy/mm/dd,hh:mm:ss* is the date and time of the beginning of the interval. The default is today at 00:00:01.
- -end *yyyy/mm/dd,hh:mm:ss* is the date and time of the end of the interval. The default is today at the current time.
- -timePeriod *minutes* is the duration of each time period, in minutes. The minimum value is five minutes. The default is 15 minutes.
- -input *fileName* is the full path name of the file that contains the audit log. The default is the file name "\$TC\_DBPATH/audit/log" (for AIX and HP-UX) or "*d*:\\%TC\_DBPATH%\audit.log" (for OS/2 or Windows NT), where *d*:\\family is the top directory for the family.

To obtain the default report of only the highest use, type the following,

```
tclicmon -highest
```

If the current date and time when the command is issued is July 31, 1996, 08:15:59 and the current directory for the family is k:\testfam, then the result shown in the standard output might be as follows:

```
*** TeamConnection License Monitor ***
```

```
Begin date:  
1996/07/31,00:00:01  
End date:  
1996/07/31,08:15:59  
Length of each time period, in minutes:  
15  
Audit file:  
K:\testfam\audit.log
```

```

The period that has the highest number of concurrent users is:
beginDate          endDate          concurrentUsers
-----
1996/07/31,07:00:00 1996/07/31,07:15:00 3

```

## Displaying a full use report

To display a full use report for an interval, issue the following command from the directory containing the family you want to monitor:

```

tclicmon -report [-outputFormat]
             [-begin yyyy/mm/dd,hh:mm:ss]
             [-end   yyyy/mm/dd,hh:mm:ss]
             [-timePeriod minutes]
             [-input fileName]

```

Where:

- **-outputFormat** is one of the following:
  - csv** Produces an output in comma-separated-values (CSV) format. This format can be used to prepare charts with software that can import data in CSV format.
    - Each row corresponds to one time period.
    - The fields are separated by a comma, and the dates are enclosed between quotes, for example:
 

```
"1996/10/01,00:00:01", "1996/10/01,14:30:15", 10
```
  - long** This is the default format for the **-report** action. Produces an output with a header and a footer, and the time periods are shown in the following table format.
    - Each field is displayed as a column heading.
    - Field values appear under respective column heading.
    - Each row corresponds to one time period.
  - raw** Produces an output in raw format:
    - Each row corresponds to one time period.
    - The fields are separated by a vertical bar, for example:
 

```
1996/10/01,00:00:01|1996/10/01,14:30:15|10
```
  - stanza** Produces an output that is equivalent to the long format.
  - table** Produces an output without a header or a footer, and the time periods are shown in the following table format:
    - Each field is displayed as a column heading.
    - Field values appear under respective column heading.
    - Each row corresponds to one time period.
- **-begin yyyy/mm/dd,hh:mm:ss** is the date and time of the beginning of the interval. The default is today at 00:00:01.
- **-end yyyy/mm/dd,hh:mm:ss** is the date and time of the end of the interval. The default is today at the current time.
- **-timePeriod minutes** is the duration of each time period, in minutes. The minimum value is five minutes. The default is 15 minutes.

- -input *fileName* is the full path name of the file that contains the audit log. The default is the file name "\$TC\_DBPATH/audit/log" (for AIX and HP-UX) or "d:\%TC\_DBPATH%\audit.log" (for OS/2 or Windows NT), where *d:\family* is the top directory for the family.

## Examples

- To obtain a default detailed report (-long) on family use, type the following command:

```
tclicmon -report -begin 1996/07/31,04:00:01
```

If the current date and time is July 31, 1996, 08:15:59, the starting time is 04:00 and the current directory for the family is k:\testfam, the result shown in the standard output might be as follows:

```
*** TeamConnection License Monitor ***
```

```
Begin date:
  1996/07/31,04:00:01
End date:
  1996/07/31,08:15:59
Length of each time period, in minutes:
  15
Audit file:
  K:\testfam\audit.log
```

beginDate	endDate	concurrentUsers
1996/07/31,04:00:01	1996/07/31,04:15:00	2
1996/07/31,04:15:00	1996/07/31,04:30:00	1
1996/07/31,04:30:00	1996/07/31,04:45:00	0
1996/07/31,04:45:00	1996/07/31,05:00:00	0
1996/07/31,05:00:00	1996/07/31,05:15:00	0
1996/07/31,05:15:00	1996/07/31,05:30:00	0
1996/07/31,05:30:00	1996/07/31,05:45:00	0
1996/07/31,05:45:00	1996/07/31,06:00:00	0
1996/07/31,06:00:00	1996/07/31,06:15:00	2
1996/07/31,06:15:00	1996/07/31,06:30:00	1
1996/07/31,06:30:00	1996/07/31,06:45:00	0
1996/07/31,06:45:00	1996/07/31,07:00:00	0
1996/07/31,07:00:00	1996/07/31,07:15:00	3
1996/07/31,07:15:00	1996/07/31,07:30:00	0
1996/07/31,07:30:00	1996/07/31,07:45:00	0
1996/07/31,07:45:00	1996/07/31,08:00:00	1
1996/07/31,08:00:00	1996/07/31,08:15:00	0

The period that has the highest number of concurrent users is:

beginDate	endDate	concurrentUsers
1996/07/31,07:00:00	1996/07/31,07:15:00	3

- To obtain a detailed report on family use in table format (without the header and footer), type the following command:

```
tclicmon -report -table -begin 1996/07/31,04:00:01
```

If the current date and time is July 31, 1996, 08:15:59, the starting time is 04:00, and the current directory for the family is k:\testfam, the result shown in the standard output might be as follows:

beginDate	endDate	concurrentUsers
1996/07/31,04:00:01	1996/07/31,04:15:00	2
1996/07/31,04:15:00	1996/07/31,04:30:00	1
1996/07/31,04:30:00	1996/07/31,04:45:00	0
1996/07/31,04:45:00	1996/07/31,05:00:00	0

```

1996/07/31,05:00:00 1996/07/31,05:15:00 0
1996/07/31,05:15:00 1996/07/31,05:30:00 0
1996/07/31,05:30:00 1996/07/31,05:45:00 0
1996/07/31,05:45:00 1996/07/31,06:00:00 0
1996/07/31,06:00:00 1996/07/31,06:15:00 2
1996/07/31,06:15:00 1996/07/31,06:30:00 1
1996/07/31,06:30:00 1996/07/31,06:45:00 0
1996/07/31,06:45:00 1996/07/31,07:00:00 0
1996/07/31,07:00:00 1996/07/31,07:15:00 3
1996/07/31,07:15:00 1996/07/31,07:30:00 0
1996/07/31,07:30:00 1996/07/31,07:45:00 0
1996/07/31,07:45:00 1996/07/31,08:00:00 1
1996/07/31,08:00:00 1996/07/31,08:15:00 0

```

- To obtain a detailed report on family use in raw format (without the header and footer), type the following command:

```
tclicmon -report -raw -begin 1996/07/31,04:00:01
```

If the current date and time is July 31, 1996, 08:15:59, the starting time is 04:00, and the current directory for the family is k:\testfam, the result shown in the standard output might be as follows:

```

1996/07/31,04:00:01|1996/07/31,04:15:00|2
1996/07/31,04:15:00|1996/07/31,04:30:00|1
1996/07/31,04:30:00|1996/07/31,04:45:00|0
1996/07/31,04:45:00|1996/07/31,05:00:00|0
1996/07/31,05:00:00|1996/07/31,05:15:00|0
1996/07/31,05:15:00|1996/07/31,05:30:00|0
1996/07/31,05:30:00|1996/07/31,05:45:00|0
1996/07/31,05:45:00|1996/07/31,06:00:00|0
1996/07/31,06:00:00|1996/07/31,06:15:00|2
1996/07/31,06:15:00|1996/07/31,06:30:00|1
1996/07/31,06:30:00|1996/07/31,06:45:00|0
1996/07/31,06:45:00|1996/07/31,07:00:00|0
1996/07/31,07:00:00|1996/07/31,07:15:00|3
1996/07/31,07:15:00|1996/07/31,07:30:00|0
1996/07/31,07:30:00|1996/07/31,07:45:00|0
1996/07/31,07:45:00|1996/07/31,08:00:00|1
1996/07/31,08:00:00|1996/07/31,08:15:00|0

```

- To obtain a detailed report on family use in comma-separated-values format (without the header and footer), type the following command:

```
tclicmon -report -csv -begin 1996/07/31,04:00:01
```

If the current date and time is July 31, 1996, 08:15:59, the starting time is 04:00, and the current directory for the family is k:\testfam, the result shown in the standard output might be as follows:

```

"1996/07/31,04:00:01","1996/07/31,04:15:00",2
"1996/07/31,04:15:00","1996/07/31,04:30:00",1
"1996/07/31,04:30:00","1996/07/31,04:45:00",0
"1996/07/31,04:45:00","1996/07/31,05:00:00",0
"1996/07/31,05:00:00","1996/07/31,05:15:00",0
"1996/07/31,05:15:00","1996/07/31,05:30:00",0
"1996/07/31,05:30:00","1996/07/31,05:45:00",0
"1996/07/31,05:45:00","1996/07/31,06:00:00",0
"1996/07/31,06:00:00","1996/07/31,06:15:00",2
"1996/07/31,06:15:00","1996/07/31,06:30:00",1
"1996/07/31,06:30:00","1996/07/31,06:45:00",0
"1996/07/31,06:45:00","1996/07/31,07:00:00",0
"1996/07/31,07:00:00","1996/07/31,07:15:00",3
"1996/07/31,07:15:00","1996/07/31,07:30:00",0
"1996/07/31,07:30:00","1996/07/31,07:45:00",0
"1996/07/31,07:45:00","1996/07/31,08:00:00",1
"1996/07/31,08:00:00","1996/07/31,08:15:00",0

```

---

## Using the server daemon monitor

The TeamConnection server daemon monitor is a real-time program that permits you to monitor the activity of the TeamConnection server daemons. It makes use of the server's shared memory space. Each TeamConnection daemon, as well as the monitor itself, attaches to the same shared memory segment. Each time a TeamConnection server daemon services a request, the shared memory segment for that particular server daemon is updated with information regarding the user who has requested the work and the nature of the request.

You can use the server daemon monitor in a number of ways:

- To determine the activity of the server
- To determine which users issue time-consuming requests
- To determine the total number of requests serviced by the TeamConnection server and the number serviced by each server daemon since it was started
- To determine if there is a problem with one or more of the server daemons

## Using the monitor command

To start the server daemon monitor, you must be logged on to the family you want to monitor on the TeamConnection server machine. The server daemon monitor program is located in the *teamcInstallPath*\bin directory. To start the server daemon monitor, issue the following command:

```
monitor refreshInterval [width]
```

### Where:

- **refreshInterval** indicates the time in seconds between successive screen updates. If you start the monitor with a refresh interval of 2, for example, then the activity monitor screen is updated with new information every 2 seconds. A refresh interval of 0 shows a snapshot of the monitor screen that is not updated. Set the refresh interval to a number low enough to capture requests as they are issued and processed. If you set the refresh interval too high, you may never see any activity occurring because the server daemon would have received and processed the request before the screen is updated. A refresh interval of 1 or 2 seconds is usually sufficient.
- **width** indicates the number of characters of status information to display for each TeamConnection server daemon. The default is 132 characters. The maximum is also 132 characters.

After you issue this command, an activity monitor screen displays, showing which server daemons are running and which are servicing requests. To exit the server daemon monitor, press any key.

The following is an example of a TeamConnection server daemon monitor screen showing 4 TeamConnection server daemons running. Two of these daemons are servicing requests.

Put sample here

- The first line shows that all four of the TeamConnection server daemons are running and that the monitor and the server daemons are using 2248 bytes of shared memory.
- The second line indicates the total number of requests serviced by the server daemons since it was last started.

- The remaining lines show one status line for each server daemon. The status lines consist of comma-separated columns showing the following information for each daemon. If a daemon is not currently servicing requests, then only the first three columns of information are displayed. The amount of information displayed is also controlled by the *width* option specified with the *monitor* command. If you issue the *monitor* command without the *width* option, 132 characters of information are shown.

#### Column number

#### Information displayed

- |    |  |
|----|--|
| 1  | Daemon index. The index number of the TeamConnection server daemon in the shared memory segment. If a server daemon is stopped normally while the server daemon monitor is running, then -- appears in this column instead of a process ID. If a server daemon is stopped abruptly or abnormally while the server daemon monitor is running, then >> appears in this column. In either case, the information about the request that was being processed when the daemon was stopped remains on the screen. After a daemon is started again, its process ID appears in this column. |
| 2  | Daemon process ID. The process ID of the TeamConnection server daemon.   |
| 3  | The number of requests serviced by the daemon since it was started.  |
| 4  | The date the last request to the server daemon was issued. The format is mm/dd/yy.   |
| 5  | The time the last request to the server daemon was issued. The format is hh:mm:ss.   |
| 6  | The TeamConnection request that is being serviced.   |
| 7  | The TeamConnection user ID that issued the request.  |
| 8  | The login ID of the TeamConnection user who issued the request.  |
| 9  | The hostname of the machine from which the request was issued.   |
| 10 | Additional information about the request being serviced. This can include, for example, details about a TeamConnection query.  |

## Monitoring the activity of the server daemons

You can use the TeamConnection server daemon monitor to determine if you have enough server daemons running for a family:

- If you find that all daemons are constantly in use, then you may need to increase the number of daemons you start when you start the family server. Each TeamConnection family server daemon can process only one request at a time. If all daemons are busy processing requests, new requests are rejected. Users whose requests are rejected receive a message like the following:

```
0010-250
```

```
A connection cannot be established with family or port <value> at
node <value> on port <value>. The error is: <value>.
```

```
To solve the problem, contact the system administrator or the
family administrator.
```

Usually a request can be processed very quickly, but some requests can take several seconds to complete if the information being requested is lengthy or the action is complex, as in a driver -commit request. If your server is having

trouble processing requests, you may want to stop the server and then restart it with more daemons, provided your license agreement permits you to do so.

- If you find that one or more daemons are rarely used, then you may need to decrease the number of daemons you start when you start the family server.

## Detecting time-consuming requests

If you notice that a specific request of a server daemon takes a long time to complete, then you can cancel the request by recycling the daemon. To recycle a server daemon, issue the following command, replacing *pid* with the process ID of the daemon.

```
kill -1 pid
```

See [for more information on recycling server daemons](#).

## Monitoring server daemon problems

Column 3 of the server daemon monitor screen displays the number of requests serviced by each server daemon. Requests should be nearly evenly distributed among the daemons. If one or more daemons shows an unusually low number of requests processed, then there may be a problem with that daemon. There can be one or more reasons for a low processing rate for a daemon:

- A request can take a long time to process. Actions such as `driver -commit`, `driver -check -long`, `driver -extract`, `release -extract`, and `report` can be time consuming.
- A request may be held pending the release of a lock on a database table. Certain actions, such as `driver -commit`, need to lock some of the database tables so that other users do not damage the data integrity before the request completes. If a database table is locked and an update request for that table is received, then the request will be held until the database table is unlocked. An update request is any request that alters the contents of the information in a database table. Requests that query the contents of a locked database table can still be completed.





---

## Part 5. Installing and working with build servers

<b>Chapter 19. Basic build concepts</b>	259
The physical structure of the build function	259
The build object model.	261
Parent-child relationships in a build tree	262
Working with a build tree	263
Putting the pieces together	265
 <b>Chapter 20. Installing the build function</b>	267
Preparing to install	269
Installing build agents and processors on OS/2 and Windows NT	270
Step 1: Starting the installation.	270
Step 2: Selecting installation options	270
Step 3: Selecting the components for installation	270
Step 4: Completing the installation	271
Setting the environment variables.	271
Environment variables for a build agent	271
Environment variables for an OS/2 build processor	272
Installing build agents and processors on AIX and on HP-UX	272
Installing build agents or processors on Windows NT	272
Creating a build processor on MVS	272
Customizing your JCL to use the LE/370 runtime libraries.	274
Environment variables for an MVS build processor	275
 <b>Chapter 21. Working with build scripts and builders</b>	277
Creating a builder	277
Writing a build script	281
Passing parameters to a build script.	281
Writing a simple build script	282
Writing an executable file for a build script	283
Testing a build script	284
Modifying the contents of a build script.	284
Putting a builder to work	285
Removing a builder from a part	286
Working with VisualAge C++ and Templates	286
 <b>Chapter 22. Working with MVS build scripts and builders</b>	287
Creating a builder for MVS builds.	287
Writing an MVS build script	291
File name conversions for MVS	291
Passing parameters to an MVS build script	292
TeamConnection syntax for MVS build scripts	293
Supported JCL syntax	293
EXEC statement	294
DD STATEMENT.	294
Example of a build script for a C compile	295
Example of a build script for a COBOL compile	297
Example of a build script for a link	298
 <b>Chapter 23. Working with parsers</b>	301
Creating a parser	301
Putting a parser to work	303
Removing a parser from a part	304
Writing a parser command file	305

<b>Chapter 24. Building an application: an example</b>	307
Starting the build processors and build agents	308
Creating builders and parsers	309
Creating the build tree for the application	309
Starting the build on the client	313
Determining the build scope.	315
Adding the job to the job queue	317
Picking up the work orders	317
Putting the build processors to work.	317
Putting the build scripts to work	317
Finishing the job and reporting the results to the user	318
Monitoring the progress of a build	318
Running a build in spite of errors	319
Building all parts, regardless of build times	319
Finding out which parts will be built	320
Canceling a build.	320
More sample build trees	321
Defining multiple outputs from a single build event	321
Synchronizing the build of unrelated parts	322

This section tells how to install and use the TeamConnection build function.

---

## Chapter 19. Basic build concepts

This chapter defines terms and briefly describes the TeamConnection pieces that work together in building an application. For more details, continue to the other chapters in this section.

The TeamConnection build function has numerous features:

- It builds applications for platforms in addition to those it runs on. Currently you can build applications using TeamConnection on the following platforms: AIX, HP-UX, MVS, OS/2, Windows NT, and Windows 95.
- Its graphical representation of the structure of an application makes it easier to visualize and change.
- It lets you build an application using any number of machines working in parallel.
- Because it is fully integrated with TeamConnection's version control system, it ensures that the correct versions of parts are used in a build.
- It can work not only with parts that represent files, such as C source files, but also with parts that represent objects, such as VisualAge Generator applications.
- It can manage other steps related to software packaging and distribution.

For more information, see "Part 6. Using TeamConnection to package products" on page 323 .

---

### The physical structure of the build function

Figure 36 on page 260 shows the structure of TeamConnection:

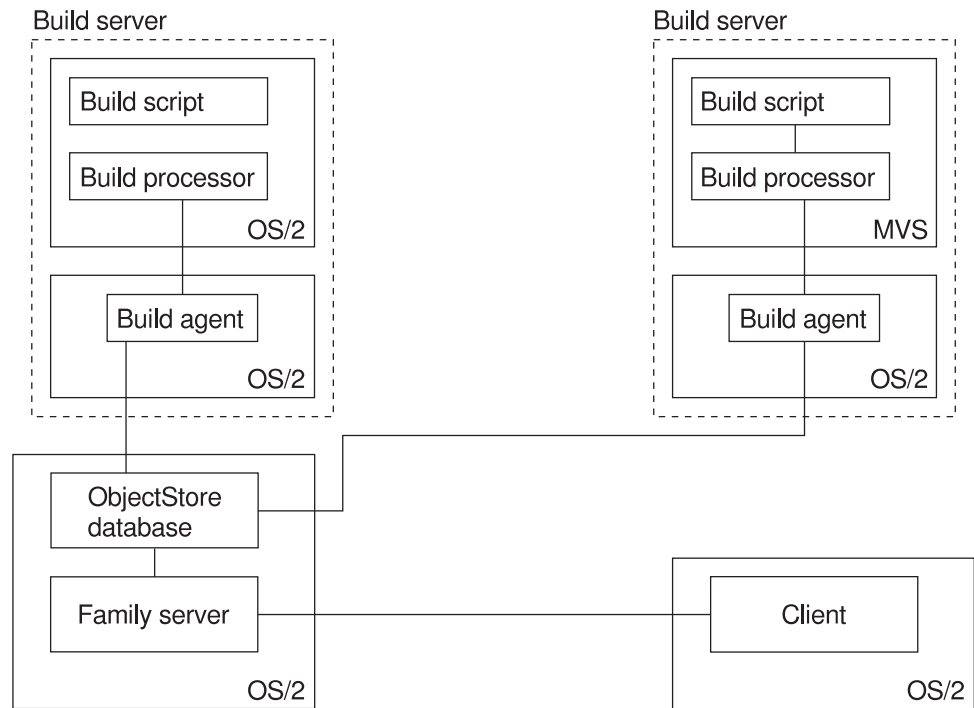


Figure 36. The physical structure of TeamConnection

Of special note to the build function are the build agents and build processors:

#### Build agent

This program handles access to parts data on behalf of the build processor. Like the family server, the build agent uses an ObjectStore database client.

There can be any number of build agents. Each is connected to one and only one build processor. TeamConnection provides build agents for the following platforms: AIX, HP-UX, OS/2, and Windows NT.

#### Build processor

This program invokes the tools, such as compilers and linkers, that construct an application. The build processor uses a build script to invoke the tools. It maintains a file cache to reduce file transfer overhead.

There can be any number of build processors. Each is connected to one and only one build agent. TeamConnection provides build processors on the following platforms: AIX, HP-UX, OS/2, Windows NT, Windows 95, and MVS. The term *build server* refers to this combination of build agent and build processor.

Build processors and agents are started by a TeamConnection administrator. For more information, see "Chapter 9. Starting and stopping the servers" on page 85.

Figure 36 shows each part of the build function on a separate machine. However, you could install them in other combinations. For more information,

---

## The build object model

Figure 37 on page 263 shows the TeamConnection objects and events that constitute the build function, as illustrated in a sample application named msgcat.exe. This build object model is a conceptual model of the build function. When you use TeamConnection to define a build, you work with a *build tree* (a simplified graphical illustration of the build object model), which you can access through the TeamConnection GUI. “Working with a build tree” on page 263 explains build trees. This section explains the build objects and events represented in a build tree.

In TeamConnection, the build function is always described and discussed in terms of the final output of the build: the product or executable file that the build produces. For the sample application shown in this illustration, msgcat.exe is the build output and appears at the top of the build object model and as the top branch of the build tree illustrated on page 263. When you want to actually build the product, you request a build of msgcat.exe. TeamConnection uses the build tree that you define for this product to determine which objects and build events it needs to generate the final output. The objects and events that TeamConnection uses for a build include the following:

### TeamConnection part

An object produced or used during a build, containing any data produced or used by the build. For example, a part called hello.c contains the source code for the application called msgcat. A part might be a text or binary file, or an object such as a VisualAge Generator generic collector.

### Build event

An individual step in the build of an application, such as the compiling of hello.c into hello.obj.

A *build scope* is a collection of build events that implement a specific build request. For example, if you start a build of an entire application, TeamConnection creates a build scope containing many build events such as compiles and links.

A *job queue* is a queue of build scopes. One job queue exists for each TeamConnection family. When a developer starts a build, the resulting build scope is added to the job queue for the family. The build agents then process the build events in the build scope, on more than one machine at a time if possible.

Build events, build scopes, and job queues are internal to TeamConnection; you cannot interact with them directly.

### Builder

An object that can transform a build event’s input parts into output parts by calling tools such as linkers or compilers. For example, one builder might know how to transform the input part hello.c into the output part hello.obj. A different builder might know how to transform hello.obj into msgcat.exe. Builders are associated with the parent, or output part, rather than the child, or input.

### Build script

An object that a builder uses in transforming inputs to outputs; it is essentially a binding between TeamConnection and a transformation tool, such as a linker or compiler. In OS/2, Windows, or UNIX environments, a build script is usually a command file, but it can be a string that calls the tool. In MVS, it is a file containing JCL.

## Parser

A tool that can read a source file and report back a list of dependencies of that source file. It frees a developer from knowing the dependencies one part has on other parts to ensure a complete build is performed. For example, a C parser can read a C source code file and report back a list of the files included by the source file or by the included files.

---

## Parent-child relationships in a build tree

One relationship that is important to understand and distinguish is the relationship between parent and child parts in a build tree.

Though parent-child relationships usually imply that the parent part generates the child part, in a TeamConnection build it is the opposite. Because TeamConnection places the build output at the top of the tree, it refers to the build output as the parent and to the build input as the child.

A child part can be related to a parent part one of three ways: it can be an input part, an output part, or a dependent part.

### Input parts

A part used as direct input to your build. An example of this is a C language source part. If you start a build and this part has changed, the changed part will be part of the new build.

### Output parts

A generated output from a build, such as an OBJ or EXE part, or a part with no contents that serves as an organizer object. If you start a build and this part has changed, the changed part will be included in the new build.

### Dependent parts

A part needed for the build operation to complete but that is not passed directly to the compiler. An example of this is an include part. If you start a build and this part has changed, the changed part will be included in the new build.

Though parent-child relationships usually imply that the parent part generates the child part, in a TeamConnection build it is the opposite. Because TeamConnection places the build output at the top of the tree, it refers to the build output as the parent and to the build input as the child.

To understand how build output is generated, it may be easier to start at the bottom of the build object model and work your way up. In Figure 37 on page 263, hello.h and bye.h are C source files that are embedded in hello.c and bye.c, respectively. The parser, parser1, is able to read hello.c and bye.c to determine files they embed. This build object model contains three build events:

- The builder compiler1 compiles hello.c into hello.obj.
- The builder compiler1 compiles bye.c into bye.obj.
- The builder linker1 links hello.obj and bye.obj into msgcat.exe

This build object model contains the following parent-child relationships:

- msgcat.exe is the parent of hello.obj and bye.obj.
- hello.obj is the parent of hello.c
- bye.obj is the parent of bye.c

You establish these parent-child relationships between parts when you create the parts in TeamConnection.

Before you can build msgcat.exe, for example, you need to create a place-holder part for it and designate linker1 as its builder. You then create place-holder parts for hello.obj and bye.obj and designate compiler1 as their builder and msgcat.exe as their parent.

walks you through an example of creating the build tree for this object model.

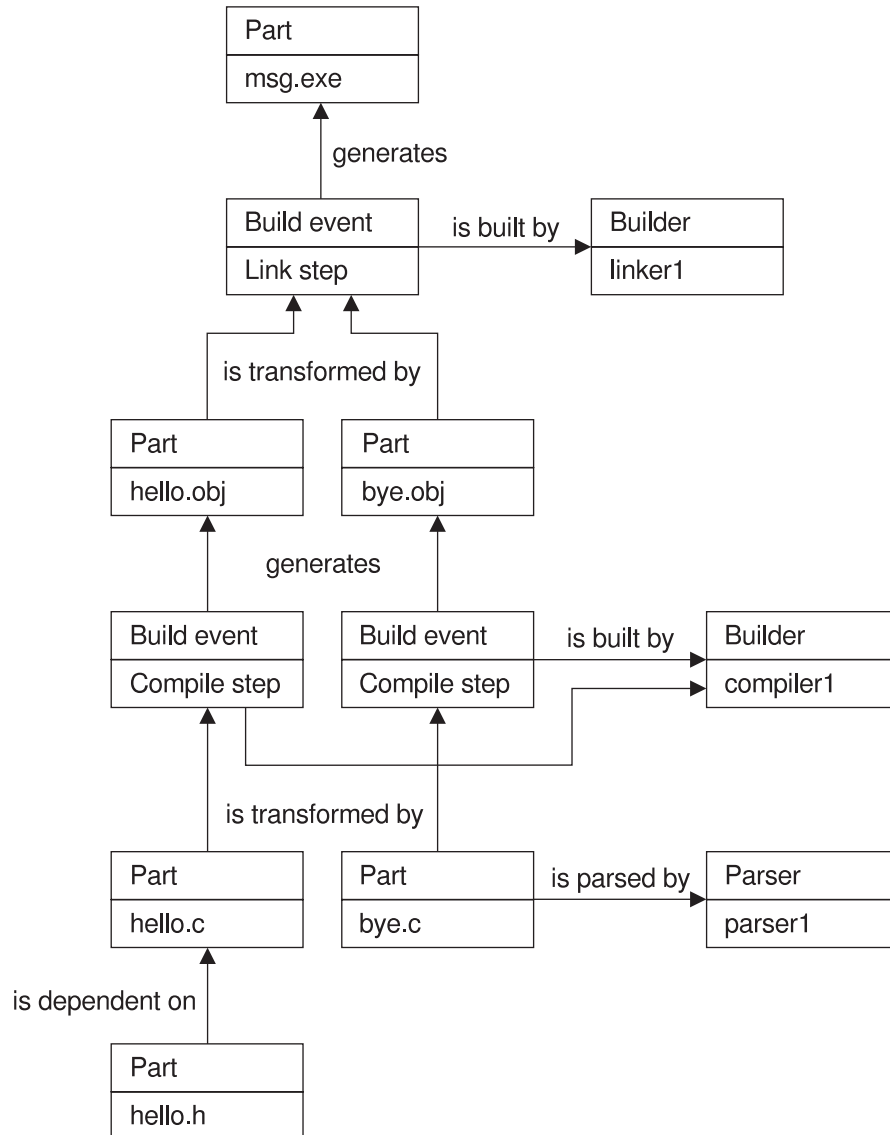


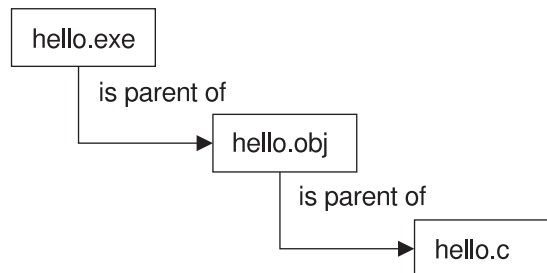
Figure 37. Sample build object model for msgcat.exe

---

## Working with a build tree

Software developers must provide the information by which TeamConnection determines the build events that make up a given build scope. An application's build tree shows this information graphically.

A build tree is a simplified version of the build object model, showing the dependencies that the parts in an application have on one another. If you change the relationship of one part to another, the build tree changes accordingly. Figure 38 shows a build tree for the hello application:



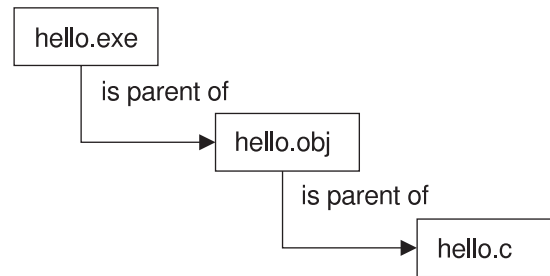
*Figure 38. The build tree for the hello application*

In this simple application, `hello.c` is compiled to create `hello.obj`, which in turn is linked to create `hello.exe`. The build tree shows that `hello.exe` is the parent of `hello.obj`, which in turn is the parent of `hello.c`. To build the entire application, you request to build `hello.exe`.

Just as the parts that make up an application are versioned, the relationships between these parts are versioned. Thus, more than one version of the build tree can exist. For example, Figure 39 on page 265 shows two different versions of the same build tree:



Work area 817



Work area 915

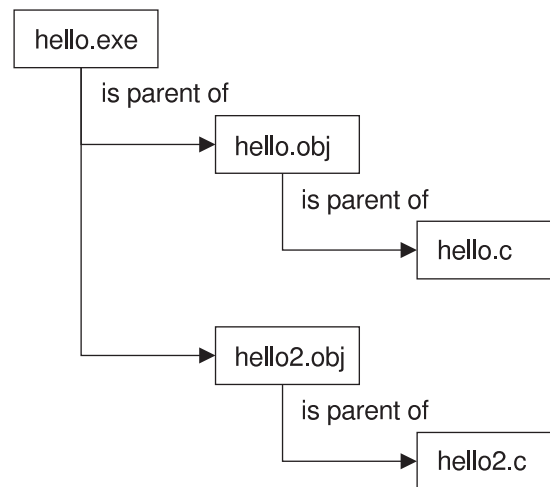


Figure 39. Two versions of a build tree

---

## Putting the pieces together

The table that follows lists the tasks involved in preparing for building an application and in actually building it. Usually an administrator does the preparation steps, but anyone with the proper authority can do so.

For more information about this task,	Go to this page.
Creating build startup files	"Creating build startup files" on page 94
Starting build servers	"Starting the servers" on page 86
Stopping build servers	"Stopping the servers" on page 95
Writing a build script	281
Creating a builder	277
Creating a parser	301



---

## Chapter 20. Installing the build function

This chapter explains what you need to do before installing build processors and build agents and provides step-by-step instructions for their installation. We assume that someone has already installed the TeamConnection family server; this chapter tells how to install a build processor or build agent on a different machine.

This chapter is intended for the build administrator for your organization.

Before installing build processors and build agents, read about the build concepts in the *User's Guide*

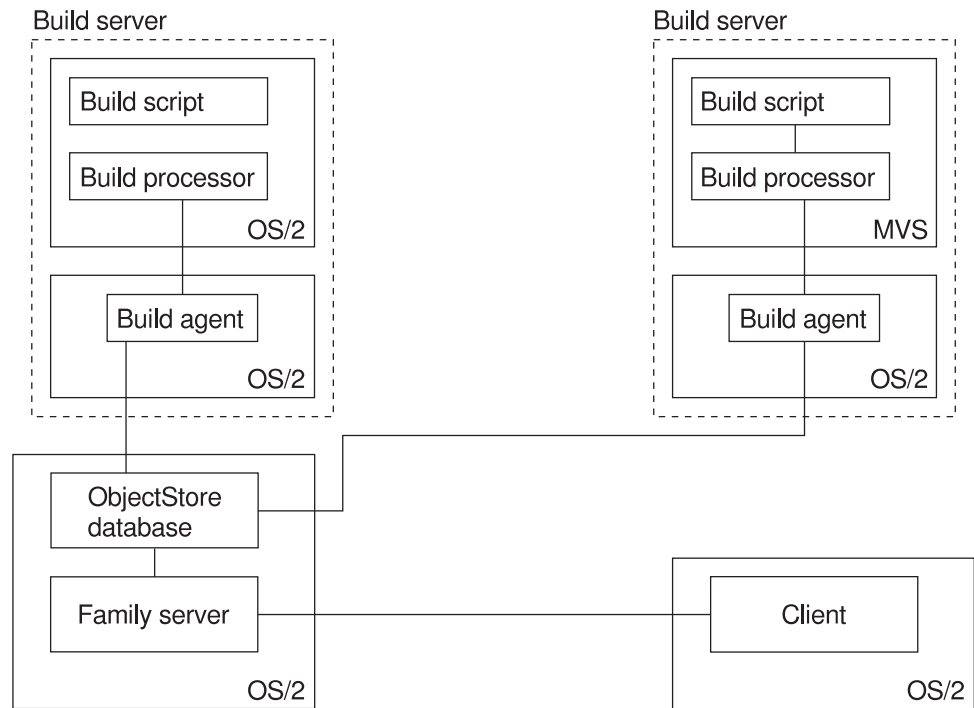
The following is a list of the tasks you do to install TeamConnection build servers:

For more information about this task,	Go to this page:
Configuring TCP/IP to prepare for installation	269
Installing build agents and processors on OS/2	270
Setting OS/2 environment variables	271
Installing build agents and processors on AIX and on HP	272
Installing an MVS build processor	272
Installing build agents and processors on Windows NT	272

For hardware requirements for the build processor and build agent machines, refer to the requirements for your specific operating system:

- For AIX, go to page 20
- For HP, go to page 32
- For OS/2, go to page 55
- For Windows NT, go to page 69

When you install the various parts of TeamConnection, you can choose to group them on a single server machine, or you can distribute them in various combinations. For example, Figure 40 on page 268 is an OS/2 configuration that shows each part on a different machine:



*Figure 40. Build agents and processors on separate machines*

Of course, you do not need to distribute the parts this way. You can run any or all of the OS/2 parts on the same machine.

For example, you could group all the build agents on one machine, producing a simplified topology that looks like Figure 41 on page 269.

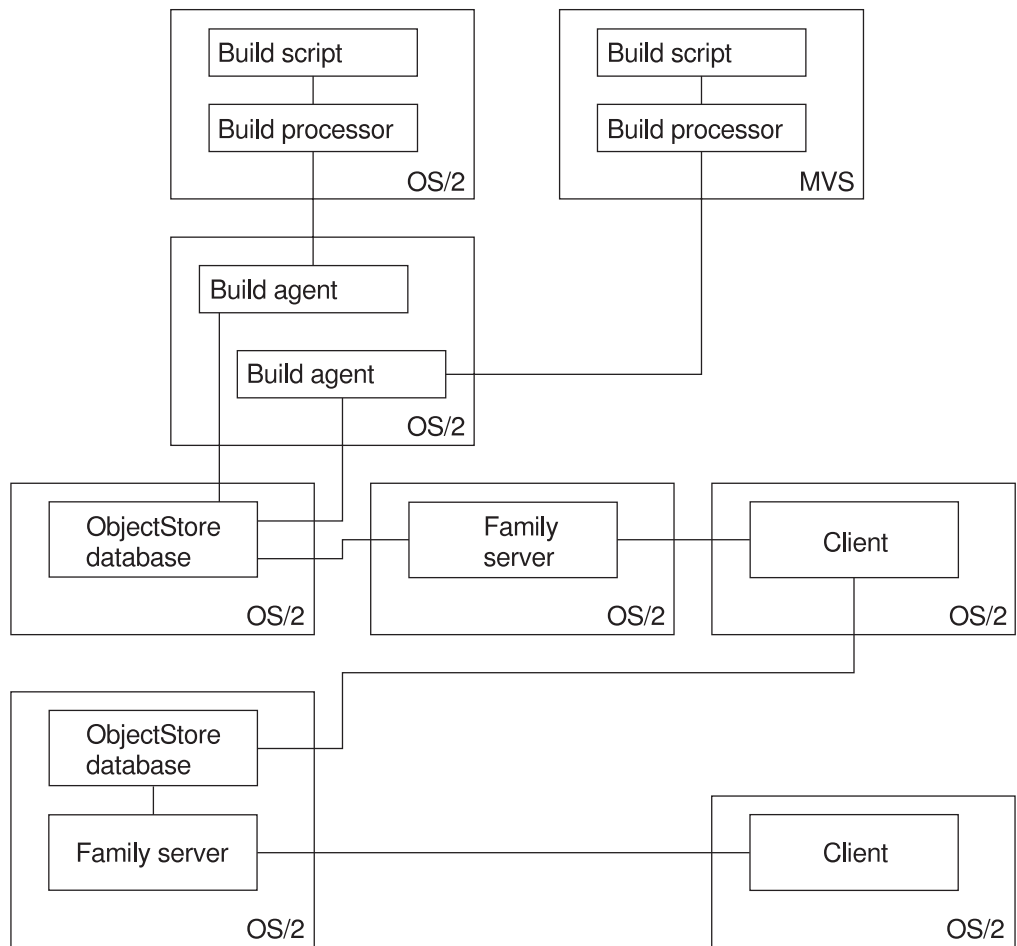


Figure 41. Build agents on a single machine

## Preparing to install

Before installing build servers, update your TCP/IP services and hosts files.

Do the following steps. For more detail about these steps, see the appropriate installation chapter for your operating system.

1. Update the services file.

**Note:** This step is optional, so long as you provide the port address when you start your build agents and processors.

The following is an example of the entry you would type in your services file:

```
# TeamConnection build servers
bldsock1 4322/tcp # port address for the build server
```

**Note:** Follow the line with a carriage return.

Remember that the port address in the services file must be the same for both the build agent machine and its matching build processor machine.

2. If you are installing a build agent, update the TCP/IP hosts file.

The following is an example of the entry you would type in your hosts file:

9.12.345.67 bldprc1.company.com bldsock1

**Note:** Follow the line with a carriage return.

3. Use the ping command to verify that you can connect to the build processor.  
Do not install the build processor or agent until the command successfully completes.

---

## Installing build agents and processors on OS/2 and Windows NT

TeamConnection provides a program for installing all or part of TeamConnection on your machine. You can use this program to install a build agent or build processor, with or without other TeamConnection components.

To install a build processor for MVS, you must first install the code on OS/2 and then upload it to MVS. Follow the instructions in this section for installing on OS/2; then go to "Creating a build processor on MVS" on page 272.

If you install a build agent on a machine that does not have a TeamConnection family server, the appropriate ObjectStore code is installed with the build agent.

This section provides instructions for installing only the build components. For information about the overall installation, refer to "Installing TeamConnection using SmartGuides" on page 59.

The time required for installing this program varies depending on your workstation. In general, installation of the product takes about 15 minutes.

### Step 1: Starting the installation

1. Insert the CD-ROM into the reader.
2. Type `x:\install` at a prompt, where *x* is the CD-ROM drive letter.  
The TeamConnection Installation window appears, followed by the Installation Instructions window.
3. Select **Continue**. The Install window appears.

### Step 2: Selecting installation options

1. From the Install window, indicate that you want to have your config.sys file updated. This is the default. If you want existing installation files overwritten during the installation, select **Overwrite files**.  
If you select not to have your config.sys file updated automatically, you will need to update the file manually after the installation program completes.  
TeamConnection creates a file called config.add that contains the environment variables that you will need to add.
2. Select **OK** or press Enter. The Install - directories window appears.

### Step 3: Selecting the components for installation

Use the Install - directories window to select which components you want to install and where you want them installed.

1. Select the components that you want to install.
2. To verify that the default drive has sufficient space, or to see what other drives are available, select **Disk space**.

If you want to use a different drive, do the following from the Disk space window:

- a. Select the drive that you want to use.
  - b. Select **Change directories to selected drive**.
  - c. Select **OK** or press Enter. The Install - directories window appears.
3. Select **Install** from the Install - directories window to start the installation. The Install - progress window shows you the progress of the installation.

## Step 4: Completing the installation

1. After the installation completes, the Installation and Maintenance window appears with an indication that TeamConnection successfully installed. Select **OK** or press Enter to return to the TeamConnection Installation window.
  2. Select **Exit** or press F3.
  3. Verify that the installation program created a folder called **TeamConnection Group**, which includes several program icons.
  4. If TeamConnection automatically updated your config.sys file during installation, restart your system now.
- If you need to manually add the environment variables to your config.sys file, go to "Setting the environment variables".

---

## Setting the environment variables

If you did not select to have the installation program update your config.sys file with the TeamConnection environment variables, TeamConnection created a file called config.add.

## Environment variables for a build agent

If you are installing a build agent, check your config.sys file to verify that the following environment variables are set:

```
NLSPATH=x:\teamc\bin\%N
TC_DBPATH=y:\dbpath
OS_ROOTDIR=x:\teamc
OS_NETWORK=o4netnp, o4nettcp
OS_TMPDIR=x:\teamc\temp
```

Where:

- *x* is the drive on which TeamConnection is installed.
- *teamc* is the default directory created for the server.
- *y* is the drive where the TeamConnection server is installed. This drive can be different than *x* if you are accessing the database remotely using a mount command. See page "Starting build servers using teamcbld" on page 89 for more information about accessing the database remotely.
- *dbpath* is the directory where the family database is located.

## Environment variables for an OS/2 build processor

If you are installing an OS/2 build processor, you can add the following environment variables to your config.sys file:

`TC_CACHESIZELIMIT=n`

`TC_CACHEPRUNEMETHOD=value`

Where:

- *n* is the maximum number of bytes to allow for the build processor's cache directory.
- *value* defines the order in which parts are pruned if the cache directory reaches the value of TC\_CACHESIZELIMIT. Valid values are the following:

**SIZE** Largest parts are pruned first.

**DATE** Least recently used parts are pruned first.

If you specify TC\_CACHESIZELIMIT but not TC\_PRUNEMETHOD, the default pruning method is by size.

If you are installing more than one build processor on the same machine, you might prefer to set these variables for each processor as you start it rather than in the machine's config.sys file.

For more information about the build cache, see "Caching and the build directories" on page 90.

---

## Installing build agents and processors on AIX and on HP-UX

Please refer to the readme file on the product installation CD.

---

## Installing build agents or processors on Windows NT

TeamConnection provides a program for installing all or part of TeamConnection on your machine. You can use this program to install a build agent or build processor, with or without other TeamConnection components.

If you install a build agent on a machine that does not have a TeamConnection family server, the appropriate ObjectStore code is installed with the build agent.

This section provides instructions for installing only the build components. For information about the overall installation, refer to "Installing TeamConnection using SmartGuides" on page 59.

---

## Creating a build processor on MVS

The code for an MVS build processor is shipped with TeamConnection and installed on your TeamConnection server when you install the product. To install the build processor on MVS, you create MVS data sets and then upload the TeamConnection files to these data sets. Follow these steps:



1. On your TeamConnection server, install the MVS build processor component, following the instructions in “Installing build agents and processors on OS/2 and Windows NT” on page 270 or “Installing build agents or processors on Windows NT” on page 272.
2. On MVS, create data sets with the following characteristics:
  - An object data set: LRECL=80, RECFM=FB, BLKSIZE=3120, DSORG=PO
  - A load module data set: LRECL=80, RECFM=U, BLKSIZE=3120, DSORG=PO
  - A JCL data set for the load modules: LRECL=80, RECFM=VB, BLKSIZE=3120, DSORG=PO
  - A JCL data set for the TEAMPROC JCL elements, such as MVS build scripts and samples: LRECL=80, RECFM=VB, BLKSIZE=3120, DSORG=PO
  - An environment variable data set for the EDCENV DDname in runpgm.jcl: LRECL=80, RECFM=VB, BLKSIZE=3120, DSORG=PS
3. From the mvs subdirectory where TeamConnection is installed, do the following to upload the files to MVS:
  - a. Type the following at a prompt and press Enter: `ftp hostname`. Specify your name and password, if required.
  - b. Type the following and press Enter after each:
    - `binary`
    - `cd 'data set for object code'`
    - `put fhbattch.mvs fhbattch`
    - `put fhbsvr.mvs fhbsvr`
    - `put fhbmsg.mvs fhbmsg`
    - `put fhbtclnk.mvs fhbtclnk`
    - `put getdsn.mvs getdsn`
    - `ascii`
    - `cd 'JCL data set for load module'`
    - `put fhblink.jcl fhblink`
    - `put runpgm.jcl runpgm`
    - `put runpgmt.jcl runpgmt`
    - `put fhbmenv.var 'environment variable data set'`
    - `quit`
  - c. This optional step installs the sample build scripts on MVS. It must be done from a machine that has the TeamConnection client installed. If you are doing these steps from a different machine than in the previous steps, repeat step a from the bin subdirectory where the client is installed on this machine. Otherwise, change to the bin subdirectory where the client is installed. Then, type the following statements and press Enter after each:
    - `ascii`
    - `cd 'data set for teamproc jcl'`
    - `put fhbmc.jcl fhbmc`
    - `put edcc.jcl edcc`

**Note:** The file `fhbmsg.mvs` is installed in the language subdirectory of the `\nls\msg` directory path in the TeamConnection installation directory, for example, `\nls\msg\en_us`. The remaining MVS files are installed in the `\mvs` subdirectory.

- put fhbmasm.jcl fhbmasm
  - put fhbplked.jcl fhbplked
  - put fhbcobm.jcl fhbcobm
  - put fhbmpli.jcl fhbmpli
  - put fhbtclnk.jcl fhbtclnk
  - put fhbm370.jcl fhbm370
  - put fhbm1ink.jcl fhbm1ink
  - quit
- d. From MVS, do the following:
- 1) Modify fhblink to customize the JCL to your MVS system.  
If you are using the LE/370 runtime libraries, go to “Customizing your JCL to use the LE/370 runtime libraries” for specific instructions.
  - 2) Submit this JCL to create the load modules fhbtclnk and teamproc.

## Customizing your JCL to use the LE/370 runtime libraries

To use the LE/370 runtime libraries, you must first modify two of the JCL files, fhblink and runpgm, as follows:

1. Modify the following in fhblink.jcl:

- Change CVER='EDC.' to CVER='CEE.'
- Change EDCBASE='SEDCBASE' to EDCBASE='SCEELKED'

Note that the sample code in fhblink.jcl uses SYS1.CEE and SYS1.EDC for reference. Customize the substitutable variables for CVER and EDCBASE to match your environment.

- Remove the statements that set the default values for the following symbolic parameters:

```
//COMHD='SYS1.'
//COMVER='PLI.'
//IBMBASE='SIBMBASE'
```

Alternatively, comment out the above statements and move them after the following statement:

```
//EZACMTX='SEZACMTX'
```

- Remove or comment out the following statement from the SYSLIB DDname statement:

```
//      DD DSN=&COMHD&COMVER&IBMBASE,DISP=SHR;
```

**Note:** After you make these changes, the following DDName statements are no longer concatenated to the SYSLIB DDName statement:

```
//      DD DSN=SYS1.EDC.SEDCBASE,DISP=SHR
//      DD DSN=SYS1.PLI.SIBMBASE,DISP=SHR
```

Instead, the following DDName statement is associated with the SYSLIB DDName statement:

```
//      DD DSN=SYS1.CEE.SCEELKED,DISP=SHR
```

2. Modify the following in runpgm.jcl:

```
//RUNPGM EXEC PGM=TEAMPROC,PARM='-S @4451 -U VIO',REGION=4M  
to  
//RUNPGM EXEC PGM=TEAMPROC,  
// PARM='ENVAR("_CEE_ENVFILE=DD:EDCENV")/-S @4451 -u VIO',REGION=4M  
EDCENV refers to the EDCENV DDName statement found in runpgm.jcl.
```

## Environment variables for an MVS build processor

If you are installing an MVS build processor, you can change the following environment variables that are defined in your environment variable data set:

```
TC_CACHESIZELIMIT=n  
TC_CACHEPRUNEMETHOD=value
```

Where:

- *n* is the maximum number of bytes to allow for the build processor's cache directory.
- *value* defines the order in which parts are pruned if the cache directory reaches the value of TC\_CACHESIZELIMIT. Valid values are the following:

**SIZE** Largest parts are pruned first.

**DATE** Least recently used parts are pruned first.

If you specify TC\_CACHESIZELIMIT but not TC\_PRUNEMETHOD, the default pruning method is by size.

For more information about the build cache, see "The build cache data sets" on page 91 .



---

## Chapter 21. Working with build scripts and builders

A builder is an object that can transform one set of TeamConnection parts into another by invoking tools such as compilers and linkers. For example, one builder might transform a COBOL source file into an object file. Another might transform a set of object files into an executable file. Builders use build scripts to invoke the tools that actually transform TeamConnection parts.

Usually a build administrator creates build scripts and builders, but anyone with the proper authority can do so. For more information about the required authority, see Appendix F. Authority and notification for TeamConnection actions.

This chapter tells how to create and maintain build scripts and builders. It assumes that you have read “Chapter 19. Basic build concepts” on page 259. The following table directs you to the tasks to be done:

For more information about this task,	Go to this page.
Creating a builder	277
Writing a build script	281
Testing a build script	284
Updating a builder	284
Putting a builder to work	285
Removing a builder from a part	286

---

### Creating a builder

As with most other TeamConnection operations, there are two ways you can create a builder: using the graphical user interface (GUI) or the command line interface.

To create a builder using the GUI:

1. From the Actions pull-down menu on the Tasks window, select **Builders → Create**.
2. On the Create Builder window, type the requested information.

Figure 42. Create Builder window

To create a builder using the command line:

From a command line, type the `teamc builder -create` command and press Enter. The complete command syntax is the following:

```
teamc builder -create name -condition RC_expression
               -environment name
               -from script_filespec
               -script name
               -value RC_value -release name
               -family name
               [-text | -binary | -none]
               [-parameters Parameters]
               [-timeout number] [-become user_name]
               [-verbose]
```

No matter which way you create a builder, you must specify a number of attributes for it. Together with the contents of the build script and the tools you use (the compilers, linkers, and so on), the following attributes define how a transformation takes place.

### Builder

The name of the builder must be unique within a release. It can be anything you want; we recommend you establish and follow a meaningful naming convention. An example of a builder name is `c_set_2`.

### Release

This is the name of the release that contains the builder. Builders are release-specific objects. They are not versioned within a release; therefore you can have only one version of a builder at any time in a release.

To use the builder from a previous release, you can link to a part that uses it in that release. This action copies the builder to the new release. Otherwise, you must create the builder again in the new release.

### Script, File type, and Source file

These fields work together to define the build script that the builder invokes to accomplish the transformation. (The **File type** field on the GUI corresponds to `-text`, `-binary`, and `-none` in the command. The **Source file** field on the GUI corresponds to the `-from` attribute.)

- If the build script is simple enough to be expressed in one line, you can specify it in the **Script** attribute when you create the builder, and specify a file type of none. At minimum, the script must specify the name of the transformation tool. For example, to invoke the C Set/2 compiler, you might specify these values:

**File type**

none

**Script** `icc`

See “Writing a simple build script” on page 282 for more information.

- If the build script is more complex, you must first create a separate file containing it; see “Writing an executable file for a build script” on page 283 for more information about how to write it. Specify the fully qualified path name of your file as the source file, and specify the file type as text or binary. TeamConnection can also detect the file type and store it in the proper format.

When the builder is created, this source file is stored as part of the builder in the TeamConnection database; during a build, the build processor creates and runs a local version of this file. Specify the name you want for this local file in the **Script** field. For example, you might specify these values:

**File type**

text

**Script** `c_compile.cmd`

**Source file**

`c:\src\c_compile.cmd`

When this builder is created, the contents of `c:\src\c_compile.cmd` are stored in the builder. When this builder is invoked, TeamConnection creates a file named `c_compile.cmd`, writes the build script into this file, and then runs it.

- If the builder is being used to only collect a set of build objects (for example, a VisualAge Generator collector part), specify these values:

**File type**

none

**Script** `null`

This prevents the build agent from extracting input and output parts to send them to the build processor.

## Environment

This is the name of the environment supported by the builder, such as OS/2, Windows, AIX, HP-UX, or MVS. The value that you specify here can be anything you like, but it must exactly match the environment value specified in a corresponding build agent. (See “Starting build servers using `teamcbld`” on page 89 for more information.) Again, we recommend you follow a naming convention for this attribute, using values such as `os2` and `mvs`.

Figure 43 on page 280 shows how the value for environment must match the environment specified in a build agent in order for a build to take place:

teamagnt	-e environment
teamc builder	-environment Name

Figure 43. Matching environment values

### Comparison operator and RC value

Together, these two attributes make up a Boolean expression that defines the criteria used to decide whether a specific build event was successfully accomplished, when evaluated against the value returned by the build script. (The **Comparison operator** and **RC value** fields on the GUI correspond to the `-condition` and `-value` attributes in the command.)

The values allowed for **Comparison operator** are as follows:

- EQ** or **==**  
Equals
- LT** or **<**  
Less than
- LE** or **<=**  
Less than or equals
- GT** or **>**  
Greater than
- GE** or **>=**  
Greater than or equals
- NE** or **!=**  
Not equal to

**RC value** can be any positive integer. An example of a Boolean expression formed from these two attributes is `return_value LE 4`, meaning that the build event is considered a success if the build script returns a value less than or equal to four.

### Parameters

This is a string passed to the build script as its argument. If the string includes blanks, enclose the string in double quotes. For example, for a builder used for VisualAge C++ compiling, you might specify a parameter string of `"/Ss /Ge-"`. If the string includes a double quote, precede the double quote with a backslash (`\`). If the string includes a dash (`-`), precede the dash with a blank space, otherwise the string is interpreted as the start of a TeamConnection action flag.

### Timeout

This attribute specifies the number of minutes that a build server will wait for an invoked build script to return before concluding an error has occurred and stopping the build event. If this occurs, the build event fails and the build agent will make itself available to process another build event.



---

## Writing a build script

When you create a builder, you must specify a build script. The build script actually invokes the transformation tool and passes it parameters defined in the **Parameters** attribute of the builder.

## Passing parameters to a build script

There are three places where parameters can be specified that affect the outcome of a build.

### As attributes of a builder

Builder parameters are passed to the build script, after variable substitution is performed. Variables are substituted based upon the following syntax:

`$(variable_name)`

To pass parameters to your build script, specify them in the **Parameters** attribute of the builder. TeamConnection sets these variables before invoking the build script.

In UNIX environments, you need to include an escape character before the `$`: `\$(variable_name)`. The following is an example: `\$(TC_INPUT)`.

You can use the following TeamConnection environment variables:

#### **TC\_FAMILY**

The TeamConnection family.

#### **TC\_RELEASE**

The release of the parts that are being built.

#### **TC\_LOCATION**

The current directory where the build script runs.

#### **TC\_INPUT**

A list of the TeamConnection parts that are input to the object being built.

#### **TC\_INPUTTYPE**

Identifies each input type.

#### **TC\_OUTPUT**

A list of the parts that are being built in this build event.

#### **TC\_OUTPUTTYPE**

Identifies each output type. The default is file.

#### **TC\_WORKAREA**

The name of the work area in which the build is being performed.

You can define other variables. These can be set when you start the build by specifying a value for **parameters** in the part -build command (from the command line or through the GUI). These variables are set in the parameters string passed to the build script.

These variables are also used to set OS/2 environment variables before the build script is invoked.

### As attributes of a part in the build tree

Parameters that are unique to a particular part are specified on the part

-create and part -modify commands. Like the builder parameters, these parameters allow variable substitution.

When parameters are specified for a part, these parameters are used *in place* of the parameters specified for the builder. In other words, if both builder and part parameters are specified, the part parameters take precedence.

In addition, whenever parameters are specified for *any* part that is an output of a build event, they apply to *all* the outputs of that build event. For example, if a build event has two outputs, msg.exe and msg.map, then changing the part parameters to `"/Debug"` for either of the two parts has the same result. The next time the build event is processed, the `"/Debug"` parameter is used when invoking the build script that produces both msg.exe and msg.map.

You can also substitute the builder parameters into the file parameters by using the variable `$(BUILDERPARMS)`. For example, you might use the following command:

```
teamc part -build myfile.c -parameters "/Ti+ $(BUILDERPARMS)" ...
```

At build time, the parameters specified in the builder for myfile.c are substituted for `$(BUILDERPARMS)`.

#### As parameters of the part -build command

The part -build command parameters are not used the same way as the other two parameters. Instead, these parameters are used to set the values of environment variables that can be used for substitution into either the builder or part parameters. They are also set in the environment so they can be retrieved by the build script. In other words, they set up the environment used by the builder.

For example, if you issue a part -build command for msg.exe, you can specify `-parameters DEBUG=YES` and, inside of both the compile and link build scripts, retrieve the value of this variable from the environment, setting compiler or linker flags accordingly.

## 95

The Windows 95 build processor does not allow you to pass other variables to the build script.

## Writing a simple build script

This kind of build script is written into the **Script** attribute of the builder. When you create or modify the builder, you specify in this attribute the name of the transformation tool to be invoked.

For example, suppose you want to create a builder that compiles a C source file into a .exe file using IBM's VisualAge C++ compiler. You specify the following attributes for the builder:

**Build script**  
icc

## Parameters

```
"$(TC_INPUT) /Fe$(TC_OUTPUT)"
```

You can create this builder using the following command:

```
teamc builder -create c_builder -script icc -parameters "$(TC_INPUT)
/Fe$(TC_OUTPUT)"
```

If you use this builder to create hello.exe from hello.c, the command actually issued by the build processor is the following:

```
"icc hello.c /fehello.exe"
```

## Writing an executable file for a build script

Suppose you need to build a C application and you want to specify at build time whether to use debug information. To do this, you define in the builder parameters a variable called *debug* and set the variable when you start the build. In this case, you need a build script that is a separate executable file to pass the debug parameter after the variable substitution.

For a build script of this form, you first write a program or command file; this file is stored in the TeamConnection database when you create the builder. When a build is performed, this build script file is extracted from the database and run. It interprets the parameters passed to it and then invokes the actual transformation tool, such as the compiler.

Our earlier example describes a builder that compiles a C source file into a .obj file using IBM's VisualAge C++ compiler. Using this builder, you can specify at build time whether to use debug information. Here is the complete build script for such a builder, written in IBM's REXX language (it could just as easily have been written in C or COBOL).

```
/* sample C Build Script using debug flag */
parse arg parms

environ = 'OS2ENVIRONMENT'
input   = VALUE('TC_INPUT',,environ)
output  = VALUE('TC_OUTPUT',,environ)
debug   = VALUE('DEBUG',,environ)

if debug = 'YES' then
do
  parms = parms || '/Ti+'
end
icc parms '/Fo'||output input

exit result
```

**95**

**NT**

Put your text here.

Windows NT and 95 build scripts must be able to return a value for a return code. Because \*.bat command files provide little support for programming logic and cannot return a value, use a compiled executable for your build script. TeamConnection provides two sample Windows build scripts and their source files. These samples, ffbwcomp.exe and ffbwlink.exe, are C programs for the Microsoft Visual C++ compiler and linker, respectively. Because these samples are C programs, they can also be used with the OS/2 build processor with only slight modifications.

You can create the builder that invokes this build script using the following command:

```
teamc builder -create c_builder2 -script c_compile.cmd -parameters "/c"
            -from d:\teamc\c_compile.cmd
```

Where d:\teamc\c\_compile.cmd is the file to be stored in the TeamConnection database and c\_compile.cmd is the name of the local file that the build processor creates and runs during a build.

To build hello.obj using the debug option, you use the following command:

```
teamc part -build hello.obj -parameters "debug=YES"
```

The command issued by the build server is the following:

```
c_compile.cmd /c
```

In turn, the build script inspects the contents of the parameters it received in its argument list and from the environment, and it forms this command:

```
"icc /c /Ti+ /fohello.obj hello.c"
```

---

## Testing a build script

The easiest way to test a build script is to write a simple driver program that sets the environment variables that the build script will expect and then runs the script against local files.

For example, to test the example build script in "Writing an executable file for a build script" on page 283, write a program that sets the TC\_INPUT, TC\_OUTPUT, and DEBUG parameters, and then runs the command file against a local copy of hello.c. If the test is successful, the script correctly builds hello.obj in the current directory, and DEBUG is interpreted correctly.

---

## Modifying the contents of a build script

Sometimes you need to modify the contents of a build script. Remember that a build script is stored as part of the builder itself. Because builders are not versioned, you do not check them out as you would most TeamConnection parts. Instead, follow these steps:

1. Extract the builder (in which the build script is stored) from the TeamConnection database.
2. Make your changes at your workstation.
3. Store the contents back into the TeamConnection database by using the builder -modify command.

For example, to modify the build script in “Writing an executable file for a build script” on page 283, you first issue the following command:

```
teamc builder -extract c_builder2 -to d:\build\c_builder2
```

Then, you use an editor to update d:\build\c\_builder2. To move the updated build script back into TeamConnection, you issue the following command:

```
teamc builder -modify c_builder2 -from d:\build\c_builder2
```

The builder is an implied dependency for any part that uses it. Therefore, the next time you build the application that uses the modified builder, all the parts that use it get rebuilt.

---

## Putting a builder to work

For an application to use a builder, the builder must be attached to the TeamConnection parts that it will build.

For an existing part, do one of the following:

- **GUI:** From the Actions menu of the TeamConnection Tasks Window, select **Parts → Modify → Properties**. On the Modify Part Properties window, type the name of the builder in the **Builder** field.

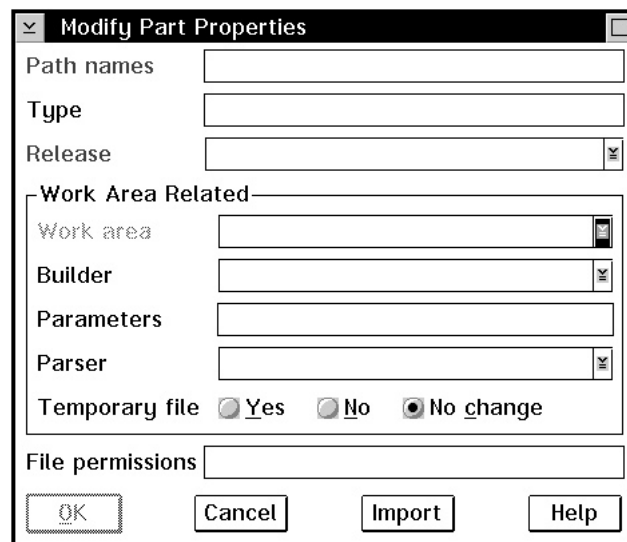


Figure 44. Modify Part Properties window

- From a command line, type the following and press Enter.

```
teamc part -modify name -Builder name
```

where the part *name* is the name of the output file to be created by this builder and the builder *name* is the name of the builder itself.

The complete syntax for this command is described in the *TeamConnection Commands Reference*.

You can also attach a builder to an output file when the part is created.

After you attach a builder to a part, the builder is ready for action. When the part is built, the builder invokes the build script, which in turn invokes a tool to transform the inputs of the part into the output.

---

## Removing a builder from a part

If you no longer want to use a builder for a part, do one of the following:

- From the GUI, select **Parts** → **Modify** → **Properties** from the Actions menu of the TeamConnection Tasks window. On the Modify Part Properties window, type null in the **Builder** field.

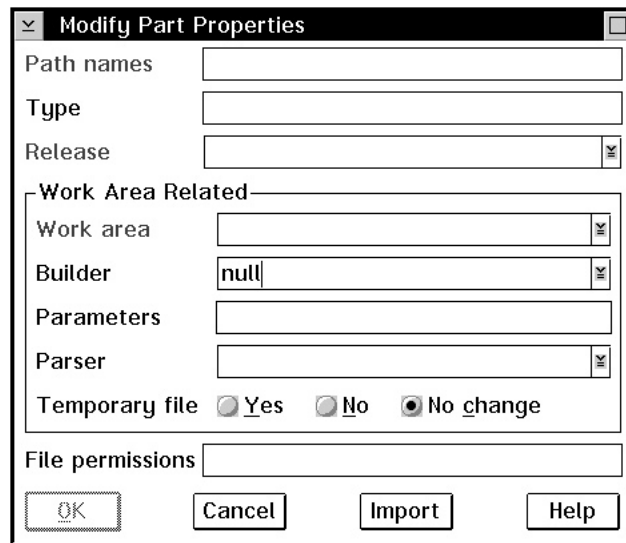


Figure 45. Modify Part Properties window

- From a command line, type the following:  
`teamc part -modify name -builder null -release name -family name`

## Working with VisualAge C++ and Templates

When using VisualAge C++ and templates, template-include objects are saved in a subdirectory of the current directory called TEMPINC, so that subsequent builds can use them. When you start a build from TeamConnection, you need to specify the /Ft(dir) parameter with your builder or use PRAGMA statements to update the template-include objects for subsequent builds. This parameter suppresses resolution of files and imbeds them within the object file.

You can specify the /Ft(dir) parameter with a builder as follows:

```
teamc builder -create c_builder -script icc -parameters "/FtE:\template"
```

---

## Chapter 22. Working with MVS build scripts and builders

A builder is an object that can transform one set of TeamConnection parts into another by invoking tools such as compilers and linkers. For example, one builder might transform a COBOL source file into an object file. Another might transform a set of object files into an executable file. Builders use build scripts to invoke the tools that actually transform TeamConnection parts.

For MVS, a build script is a text file that contains JCL statements with additional TeamConnection syntax and substitutable variables. TeamConnection parses these statements and does the necessary allocations and program calls for a build.

Usually a build administrator creates build scripts and builders, but anyone with the proper authority can do so. For more information about the required authority, see Appendix F. Authority and notification for TeamConnection actions.

This chapter tells how to create MVS build scripts and builders. It assumes that you have read “Chapter 19. Basic build concepts” on page 259. The following table directs you to the tasks to be done. In some cases, if the instructions are the same for OS/2 and MVS, the table refers you to topics in “Chapter 21. Working with build scripts and builders” on page 277.

For more information about this task,	Go to this page.
Creating a builder for MVS builds	287
Writing an MVS build script	291
Testing a build script	284
Updating a builder	284
Putting a builder to work	285
Removing a builder from a part	286

---

### Creating a builder for MVS builds

As with most other TeamConnection operations, there are two ways you can create a builder: using the graphical user interface (GUI) or the command line interface.

To create a builder using the GUI:

1. From the Actions pull-down menu on the Tasks window, select **Builders → Create**.
2. On the Create Builders window, type the requested information.

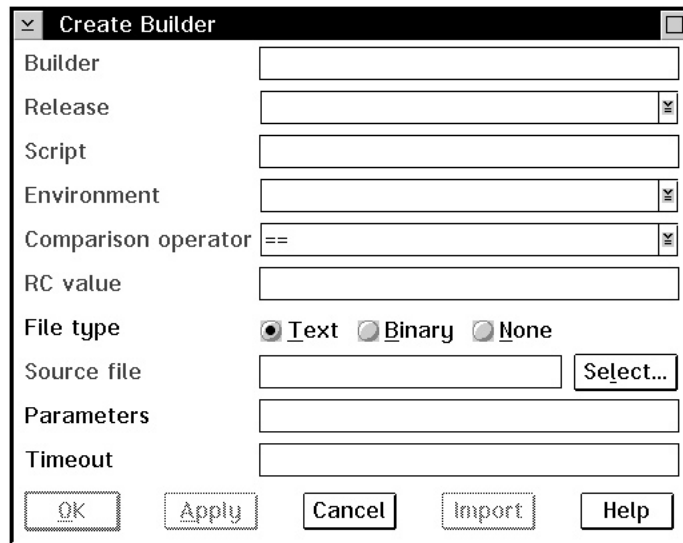


Figure 46. Create Builder window

To create a builder using the command line:

From an OS/2 command line, type the builder -create command and press Enter. The complete command syntax is the following:

```
teamc builder -create name -condition RC_expression
               -environment name
               -from script_filespec
               -script name
               -value RC_value -release name
               -family nName
               [-text | -binary | -none]
               [-parameters parameters]
               [-timeout number] [-become user_name]
               [-verbose]
```

No matter which way you create a builder, you must specify a number of attributes for it. Together with the contents of the build script and the tools you use (the compilers, linkers, and so on), the following attributes define how a transformation takes place.

### Builder

The name of the builder must be unique within a release. It can be anything you want; we recommend you establish and follow a meaningful naming convention. An example of a builder name is c370.

### Release

This is the name of the release that contains the builder. Builders are release-specific objects. They are not versioned within a release; therefore you can have only one version of a builder at any time in a release.

To use the builder from a previous release, you can link to a part that uses it in the previous release. This action copies the builder to the new release. Otherwise, you must create it again in the new release.

### Script, File type, and Source file

These fields work together to define the build script that the builder invokes to accomplish the transformation. (The **File type** field on the GUI corresponds to -text, -binary, and -none in the command. The **Source file** field on the GUI corresponds to the -from attribute in the command.)



You must first create a separate OS/2 file containing the build script. All MVS build scripts must be written using JCL statements and the TeamConnection syntax described in "Writing an MVS build script" on page 291 . You can store the build script one of two ways:

- **To store the build script as part of the builder:** specify the fully qualified path name of your build script file as the source file, and specify the file type as text. When the builder is created, this source file is stored as part of it in the TeamConnection database.

During a build, the build processor creates and runs a local version of this file. Specify the name you want for this local file in the **Script** field. For example, you might specify these values:

**File type**

text

**Script** fhbc

**Source file**

C:\build\script\fhbc.jcl

When this builder is created, the contents of C:\build\script\fhbc.jcl are stored in the builder. When this builder is invoked, TeamConnection creates a file named FHBC in the data set referenced by the TEAMPROC ddname, writes the build script into this file, and then runs it.

- **To store the build script on MVS:** create the build script file and place it in the data set allocated to the TEAMPROC ddname. When you do this, specify the following attributes:

**File type**

none

**Script** link

Do not specify a source file.

- If the builder is being used to only collect a set of build objects (for example, a VisualAge Generator collector part), specify these values:

**File type**

none

**Script** null

**Environment**

This is the name of the environment supported by the builder, such as MVS. The value that you specify here can be anything you like, but it must exactly match the environment value specified in a corresponding build agent. (See "Starting build servers using teamcbld" on page 89 for more information.) Again, we recommend you follow a naming convention for this attribute, using values such as os2 and mvs.

Figure 47 on page 290 shows how the value for environment must match the environment specified in a build agent and in the part -build command in order for a build to take place:

teamagnt	-e environment
teamc builder	-environment Name

Figure 47. Matching environment values

### Comparison operator and RC value

Together, these two attributes make up a Boolean expression that defines the criteria used to decide whether a specific build event was successfully accomplished, when evaluated against the value returned by the build script. (The **Comparison operator** and **RC value** fields on the GUI correspond to the `-condition` and `-value` attributes in the command.)

The values allowed for **Comparison operator** are as follows:

- EQ** or **==**  
Equals
- LT** or **<**  
Less than
- LE** or **<=**  
Less than or equals
- GT** or **>**  
Greater than
- GE** or **>=**  
Greater than or equals
- NE** or **!=**  
Not equal to

**RC value** can be any positive integer. An example of a Boolean expression formed from these two attributes is `return_value LE 4`, meaning that the build event is considered a success if the build script returns a value less than or equal to four.

### Parameters

This is a string passed to the build script as its argument. For example, for a builder used for linking load modules, you might specify a parameter string of `list,test`.

### Timeout

This attribute specifies the number of minutes that a build agent will wait for an invoked build script to return before concluding an error has occurred and stopping the build event. If this occurs, the build event is queued again to be processed by the next available build agent.

Because MVS builds are processed in batch mode but the build is submitted to the build agent in real time, consider writing a user exit to check the time of day before allowing a build request to be submitted. Another approach to handling the timing of MVS builds is to start the MVS build agent only at night and ensure that the MVS builders do not have short timeout values.

**Note:** The ddname TEAMPROC must be defined to a shared data set when the MVS build processor is started; see “An MVS build server” on page 91 for more information. This data set is used as a cache for the build scripts of MVS builders.

---

## Writing an MVS build script

The best starting point for an MVS build script is an existing JCL fragment that is used for transforming inputs into outputs. For example, suppose you want to create a builder that compiles a C source file into an OBJECT file using IBM's C/370 compiler. You probably already have JCL that can be submitted as a batch job that does this.

When you create a build script for the MVS environment, you specify JCL statements with additional TeamConnection syntax. This build script is parsed by the build processor. From the parsed results, TeamConnection allocates the specified ddnames and data sets; it then determines and executes the programs dynamically. The MVS build processor uses the specialized TeamConnection syntax in the JCL to determine where to store the parts involved in an MVS build.

All statements in the MVS build script except for comments and inline data stream must start with two forward slashes (/ /).

Before you start writing your build script, refer to the manuals for the compiler, linker, or other transformation program to determine the data set requirements. Pay particular attention to the DCB attributes for LRECL, BLKSIZE, and RECFM.

Sample build scripts shipped with TeamConnection can be installed on MVS. Page 476 lists the sample build scripts. For instructions on installing these samples,

If you are debugging a build script, these manuals are also the first place to look for problems.

For more information about JCL syntax, refer to the *JCL User's Guide* and *JCL Reference* for your version of MVS. (These are listed in the bibliography at the back of this book.)

The following sample MVS build scripts are shipped with TeamConnection:

**fhbmasm.jcl**

Calls the MVS assembler

**fhbcobm.jcl**

Calls the MVS COBOL compiler

**fhbmpli.jcl**

Calls the PL/1 MVS compiler

## File name conversions for MVS

TeamConnection file names are modified by the MVS build server according to the following rules:

- The directory path of a file name is not used. All characters of a file name up to and including the rightmost slash (/ or \) are thrown away.
- Lowercase characters are converted to uppercase characters.

- The file extensions are stripped from the right, up to and including the leftmost period. The extension, minus the period, is used by the MVS build tool to direct the file to particular data sets according to user-specified syntax in the MVS build scripts.

- The remaining name is truncated from the left, to a maximum of 8 characters.

- Names must contain characters that are valid in MVS. MVS allows the following characters:

0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ\$@#

However, the name must begin with an alphabetic character.

- Underscore characters (\_) in a base name are converted to at signs (@).

The following are examples of how a TeamConnection name is converted:

- A TeamConnection file name of src\build\fhbldobj.C is converted to FHBLOBJ on MVS.
- A TeamConnection file name of src/build/fhbtruncate.c is converted to FHBTRUNC on MVS.

In both of these examples, the .C or .c is split away. The MVS build processor uses the resulting C extension to resolve and possibly allocate the MVS data sets needed for the build process.

A TeamConnection file name of src\build\fhbtest.c.old is converted to FHBTEST, and c.old becomes the extension.

## Passing parameters to an MVS build script

To pass parameters to your build script, specify them in the **Parameters** attribute of the builder. These are passed to MVS through the combination of the PARM keyword parameter on an EXEC card and the &TCPARM variable.

**Note:** Take extra care to use no single or double quotes in the **Parameters** attribute of the MVS builder definition. This rule follows standard JCL syntax for parameter substitution in the PARM keyword parameter of an EXEC statement.

You can use the **&TCPARM** variable in your MVS build scripts. This variable is substituted with any parameters that were specified using the -parameter attribute of the builder command or the **Parameters** field on the Create Builder window when the builder was created.

You can also use the following TeamConnection variables in writing MVS build scripts:

### &TCINPUT

This variable is used for in-stream data. For each build input, the line where &TCINPUT appears is duplicated and the variable &TCINPUT substituted with the input name.

### &TCOUTPUT

This variable is used for in-stream data. For each build output, the line where &TCOUTPUT appears is duplicated and the variable &TCOUTPUT substituted with the output name.

### &TCWKAREA

The name of the work area in which the build is being performed.

## **&TCRELEAS**

The name of the release in which the build is being performed.

**Note:** The &TCINPUT and &TCOUTPUT substitutable variables have limited scope in the MVS build scripts and should be used only within the in-stream data.

You can define other variables. You can set them by specifying a value for **Parameters** when you start a build. These variables are set in the parameters string passed to the build script.

Further, these variables can be used for variable substitution within MVS build scripts. Variable substitution works similarly to JCL variable substitution.

## **TeamConnection syntax for MVS build scripts**

TeamConnection has extended the existing JCL syntax. The extended syntax tells the TeamConnection build processor where to put the inputs, where to get the outputs, and where to get messages from the translators after an MVS build.

To direct inputs, outputs, and messages, add TCEXT=xxx to the data set attributes defined to a ddname, where xxx is one of the following:

- The base name extension from the TeamConnection part—for example, TCEXT=H, where H is the extension from A.H.
- One or more base name extensions from TeamConnection parts, surrounded by parentheses—for example, TCEXT=(H,HPP), where H is an extension from A.H or HPP is an extension from A.HPP.
- The string TCOUT, which declares that the contents of the data set assigned to the ddname will be sent back to TeamConnection. Users can view this information in one of these ways:
  - On an OS/2 command line, typing teamc part *name* -viewmsg and pressing Enter
  - Selecting **Part → View → View build message** from the Actions pull-down menu on the Tasks window

When you add the TCEXT attribute for a ddname specification, you must also specify other attributes to allocate the data set through dynamic allocations:

- SPACE
- UNIT
- DCB, which includes the LRECL, BLKSIZE, and RECFM attributes

The UNIT attribute defaults to VIO unless the -U parameter is specified when the MVS build processor is started.

For translation messages, you can allocate a data set to the ddname TC\$LIST and specify the attributes yourself. Otherwise, the build processor allocates this data set with the following attributes by default:

```
//TC$LIST DD DCB=(RECFM=VB,LRECL=255,BLKSIZE=32640),  
// SPACE=(CYL,(2,1)),DISP=(NEW,DELETE),UNIT=VIO
```

## **Supported JCL syntax**

The TeamConnection MVS build processor supports only a subset of the available JCL syntax.

The following are not supported:

- A JOBSTEP statement
- DISP=(..,PASS)...

JCL procedures can be used on an EXEC statement. However, you must verify that any procedure called by the build script uses syntax that TeamConnection supports.

The following list indicates the positional and keyword parameters that are supported. You can verify the syntax in the *JCL Reference*.

## EXEC statement

```
//label EXEC positional_parameter,keyword_parameter
```

The following parameters are supported.

### Positional parameters:

- PGM=*program\_name*, where *program\_name* is an executable load module
- PROC=*procedure\_name*, where *procedure\_name* is an existing JCL procedure
- *procedure\_name*, where *procedure\_name* is an existing JCL procedure

### Keyword parameters:

- PARM='*information*', where *information* is the parameter string passed to the load module.
- COND=(*code,operator [,stepname]*)
  - *code* is the value to test against the return code from a previous step
  - *operator* is the comparison to be made between the value for *code* and the return code
  - *stepname* is the step issuing the return code

All other keyword parameters are ignored and not used.

## DD STATEMENT

```
//label DD keyword_parameter
```

### Positional parameters

The only supported positional parameter is *[\*]*, which begins an in-stream data set containing no JCL.

### Keyword parameters

The following keywords are supported.

- DSN=*data\_set\_name* or DSNAME=*data\_set\_name*
- DISP=*status* or DISP=(*[status] [,normal-termination-disp] [,abnormal-termination-disp]*)
  - Valid values for *status* are NEW, OLD, SHR, or MOD.
  - Valid values for *normal\_termination\_disp* or *abnormal\_termination\_disp* are DELETE, KEEP, CATLG or UNCATLG.
- UNIT=*unit\_type*, where *unit\_type* is any value allowed in JCL. The default is VIO unless a different default is set when the MVS build processor is started.
- SPACE=(*allocation\_type*,(*primary[, secondary] [,directory]*))[*,RLSE*][*,CONTIG*])

- Valid values for *allocation\_type* are TRK, CYL, or the block size.
- *primary* is the primary number of the allocation type.
- *secondary* is the secondary number of the allocation type.
- *directory* is the number of directory blocks for a partitioned data set.
- DCB=(LRECL=record\_length,BLKSIZE=block\_size,RECFM=record\_format)  
Valid values for *record\_format* are F, FB, V, VB, or U).
- DSORG=*data\_set\_organization*  
Valid values for *organization* are the following:
  - P0 for a partitioned data set
  - PS for a sequential data set
- DDNAME=*label*, where *label* is the later ddname label reference. This parameter is supported only for simple cases.
- SYSOUT=*class*  
This will always be allocated as a DUMMY DSN.

All other keyword parameters are ignored and not used.

## Example of a build script for a C compile

The following JCL can be submitted as a batch job to do the following:

- Compile the source file member in the data set WELLSK.TEAMC.C
- Produce an object file member in the data set WELLSK.TEAMC.OBJ
- Produce a listing of the source file in the file member in the data set WELLSK.TEAMC.LISTING
- List the compiler messages in the file member in the data set WELLSK.TEAMC.ERROR

```

//COMPILE EXEC PGM=EDCCOMP,
// PARM='LO,SSCOMM,NOSEQ,NOMAR,LIS,FL(I),SO,DECK,TEST',
//      REGION=1536K
//STEPLIB DD DSN=SYS1.EDC.SEDCCOMP,DISP=SHR
//      DD DSN=SYS1.EDC.SEDCLINK,DISP=SHR
//      DD DSN=SYS1.PLI.SIBMLINK,DISP=SHR
//SYMSGSGS DD DSN=SYS1.EDC.SEDCDMSG(EDCMSGE),DISP=SHR
//SYSIN DD DSN=WELLSK.TEAMC.C(MEMBER),DISP=SHR
//USERLIB DD DSN=WELLSK.TEAMC.H,DISP=SHR
//SYSLIB DD DSN=SYS1.EDC.SEDCHDRS,DISP=SHR
//SYSPUNCH DD DSN=WELLSK.TEAMC.OBJ(MEMBER),DISP=SHR
//SYSLIN DD SYSOUT=*
//SYSPRINT DD DSN=WELLSK.TEAMC.ERROR(MEMBER),DISP=SHR
//SYSPRT DD DSN=WELLSK.TEAMC.LISTING(MEMBER),DISP=SHR
//SYSUT1 DD UNIT=VIO,DISP=(NEW,DELETE),
// SPACE=(32000,(30,30)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT4 DD UNIT=VIO,DISP=(NEW,DELETE),
// SPACE=(32000,(30,30)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT5 DD UNIT=VIO,DISP=(NEW,DELETE),
// SPACE=(32000,(30,30)),DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT6 DD UNIT=VIO,DISP=(NEW,DELETE),
// SPACE=(32000,(30,30)),DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT7 DD UNIT=VIO,DISP=(NEW,DELETE),
// SPACE=(32000,(30,30)),DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT8 DD UNIT=VIO,DISP=(NEW,DELETE),
// SPACE=(32000,(30,30)),DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT9 DD UNIT=VIO,DISP=(NEW,DELETE),
// SPACE=(32000,(30,30)),DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
//SYSUT10 DD SYSOUT=*
//

```

*Figure 48. A JCL fragment for an MVS compile*

The first step in converting the JCL fragment is to recognize the intent for each of the data sets and ddnames. For this C/370 compiler example, the SYSIN ddname needs to be associated with the source file, the SYSPUNCH ddname needs to be associated with the object file, and so on.

In each of these cases, the build script must tell the TeamConnection build processor where to put or pick up the parts before and after the execution of the specified program (PGM=EDCCOMP).

Assume that your source files in TeamConnection have the extension .c, your object files have .obj, and your include files .h or .hpp. You allocate a data set to the SYSIN ddname to contain a source file with a .c extension. You specify the DCB, UNIT, DISP, and SPACE attributes to dynamically create this data set every time this build script is invoked. Notice that the attribute SPACE=(TRK,(10,5)) indicates a sequential data set organization.

You specify the output messages that will be returned to TeamConnection by using the TCOUT attribute. This attribute tells the MVS build processor to return the information in the data set associated with the TCEXT=TCOUT attribute.

**Note:** The STEPLIB is renamed by the MVS build processor to STEPLIBB for data set lookup of the program specified by the PGM parameter on an EXEC statement.



The following MVS build script is the result of converting the JCL fragment by adding the TeamConnection MVS JCL syntax.

```
//COMPILE EXEC PGM=EDCCOMP,
// PARM='LO,SSCOMM,NOSEQ,NOMAR,LIS,FL(I),SO,DECK,&TCPARM',
// REGION=1536K
//STEPLIB DD DSN=SYS1.EDC.SEDCCOMP,DISP=SHR
// DD DSN=SYS1.EDC.SEDCLINK,DISP=SHR
// DD DSN=SYS1.PLI.SIBMLINK,DISP=SHR
//SYMSGS DD DSN=SYS1.EDC.SEDCDMSG(EDCMSGE),DISP=SHR
//SYSIN DD TCEXT=(C,CPP),DISP=(NEW,DELETE),
// UNIT=SYSDA,SPACE=(TRK,(10,5)),
// DCB=(RECFM=VB,LRECL=150,BLKSIZE=3200)
//USERLIB DD TCEXT=(H,HPP),DISP=(NEW,DELETE),
// UNIT=VIO,SPACE=(TRK,(5,10,10)),
// DCB=(RECFM=VB,LRECL=50,BLKSIZE=3200)
//SYSLIB DD DSN=SYS1.EDC.SEDCHDRS,DISP=SHR
//SYSPUNCH DD TCEXT=OBJ,DISP=(NEW,DELETE),
// UNIT=VIO,SPACE=(TRK,(10,5)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSLIN DD SYSOUT=*
//SYSPRINT DD TCEXT=TCOUT,DISP=(NEW,DELETE),
// SPACE=(32000,(30,30)),UNIT=VIO,
// DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
//SYSPRT DD TCEXT=TCOUT,DISP=(NEW,DELETE),
// SPACE=(32000,(30,30)),UNIT=VIO,
// DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
//SYSUT1 DD UNIT=VIO,DISP=(NEW,DELETE),
// SPACE=(32000,(30,30)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT4 DD UNIT=VIO,DISP=(NEW,DELETE),
// SPACE=(32000,(30,30)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT5 DD UNIT=VIO,DISP=(NEW,DELETE),
// SPACE=(32000,(30,30)),DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT6 DD UNIT=VIO,DISP=(NEW,DELETE),
// SPACE=(32000,(30,30)),DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT7 DD UNIT=VIO,DISP=(NEW,DELETE),
// SPACE=(32000,(30,30)),DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT8 DD UNIT=VIO,DISP=(NEW,DELETE),
// SPACE=(32000,(30,30)),DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT9 DD UNIT=VIO,DISP=(NEW,DELETE),
// SPACE=(32000,(30,30)),DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
//SYSUT10 DD SYSOUT=*
//
```

Figure 49. A JCL fragment converted to a build script

## Example of a build script for a COBOL compile

TeamConnection provides a sample build script program for compiling MVS COBOL programs. This sample is called `fhbcobm.jcl`. It invokes a JCL procedure called `IGYWC`, which needs to be in the system `PROCLIB` concatenation or in the data set identified by the `TEAMPROC` DD statement in the MVS build processor job. You may need to adjust the default parameters for the system. The following JCL should work with any IBM COBOL/II type of compiler such as the IBM COBOL/II compiler `IGYCRCTL`:

```
/*-----
/* PROGRAM:  cobolcmp.jcl
/* IBM COBOL for MVS
/* Compile Only
/*
/*-----
```

```

//COBOLCMP EXEC PGM=IGYCRCTL,PARM='&TCPARM'
//*
//* INPUT FILES WITH EXTENSION CBL
//*
//SYSIN DD TCEXT=CBL,DISP=(NEW,DELETE),
// SPACE=(32000,(30,10)),UNIT=SYSDA,
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
//*
//* COPY FILES WITH EXTENSION CPY
//*
//SYSLIB DD TCEXT=CPY,DISP=(NEW,KEEP),
// SPACE=(32000,(30,30,30)),UNIT=SYSDA,
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
//*
//SYSPRINT DD TCEXT=TCOUT,DISP=(NEW,DELETE),
// SPACE=(32000,(30,30)),UNIT=SYSDA,
// DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3990)
//SYSLIN DD TCEXT=OBJ,UNIT=SYSDA,
// DISP=(NEW,DELETE),SPACE=(32000,(30,10)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//*
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT6 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//

```

## Example of a build script for a link

Because MVS load modules are not easily transferable, TeamConnection provides a sample build script program that reads linkage editor SYSLIN control statements. This script produces a single file that can be returned from MVS and loaded into TeamConnection. You can later extract the file and transport it to MVS, where it can be link edited to produce an executable load module.

The next example shows this sample build script, named fhbtclnk.jcl, which is shipped with the TeamConnection client.

You can use either of the following for an INCLUDE control statement for the FHBTCCLNK program:

- INCLUDE DDNAME(MEMBER)
- INCLUDE DDNAME

This syntax is a subset of the linkage editor INCLUDE card.

If the card is an INCLUDE ddname(MEMBER) control statement, the object code is copied into a sequential data set associated with the SYSMOD ddname. Otherwise, the control card is embedded in the data set associated with the SYSMOD ddname. This data set can be returned as the output from this build script.

```

//FHBTCCLNK EXEC PGM=FHBTCCLNK,
// PARM='SIZE=(768K,192K),LIST,MAP,AMODE(31),RMODE(24),LET,XREF'
//STEPLIB DD DSN=userid.teamc.LOADLIB,DISP=SHR
//SYSMOD DD TCEXT=LOAD,DISP=(NEW,DELETE),
// SPACE=(32000,(30,10)),UNIT=VIO,
// DCB=(RECFM=U,LRE10CL=80,BLKSIZE=3200)
//OBJ DD TCEXT=(OBJ,PRE),DISP=(NEW,DELETE),
// UNIT=VIO,SPACE=(32000,(30,10,10)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)

```

```
//SYSPRINT DD TCEXT=TCOUT,DISP=(NEW,DELETE),
// UNIT=VIO,SPACE=(TRK,(30,10)),
// DCB=(RECFM=FB,LRECL=121,BLKSIZE=1210)
//SYSLIN DD *
INCLUDE OBJ(&TCINPUT)
ENTRY CEESTART
//
```

TCEXT attributes have been added to the following DD statements:

**Data set**

**Purpose**

**SYSMOD**

Return the output to check in to TeamConnection

**OBJ** Receive the object files transported to MVS from TeamConnection

**SYSPRINT**

Return any FHBTCLNK messages to TeamConnection

In the SYSLIN data stream, the statement INCLUDE OBJ(&TCINPUT) will be duplicated for all of the inputs to this build. The &TCINPUT variable will be replaced with the base name of the input without the extension.

To use the output of this build script as an MVS executable, do the following:

1. Extract the output from TeamConnection.
2. Transfer the output as a binary file from your workstation to MVS (for example, using FTP).
3. Link edit this output into a load module. Possible SYSLIN control statements for the link step include the following:

```
//SYSLIN DD *
INCLUDE OBJECT(OUTPUT)
NAME module(R)
//
```

The output specified in INCLUDE OBJECT(OUTPUT) contains embedded control statements specified from the build script FHBTCLNK. The linkage editor recognizes these embedded statements and produces an executable load module from the output file. The NAME control statement cannot be embedded in the output data set.



---

## Chapter 23. Working with parsers

This chapter describes how to create a parser. It assumes that you have read "Chapter 19. Basic build concepts" on page 259.

Consider the task of defining and maintaining a build tree. One of the more time-consuming, and error-prone, portions of this task is defining the dependencies that one TeamConnection part has on others.

For example, if hello.c includes hello.h, you need to define hello.h as a dependency of hello.c in the build tree. That sounds simple enough, but imagine a real application in which there are hundreds of dependencies and the dependencies have dependencies. Defining such a tree becomes very difficult; maintaining it, even more so.

To solve this problem and automate some of the work of defining and maintaining a build tree, you can instead use a parser object. The task of a parser is to inspect source code to determine dependencies. In the previous example, a parser can inspect hello.c, recognize that it has a dependency on hello.h, and create that dependency in the TeamConnection build tree.

Because parsers are language-dependent, you probably need a different parser for each language you use in a particular release. For example, you might have both a COBOL parser and a C parser in a release. Many parts in the release can use the same parser.

Usually a TeamConnection administrator defines parsers, but anyone with the proper authority can do so. For more information about the required authority, see Appendix F. Authority and notification for TeamConnection actions.

---

### Creating a parser

As with most other TeamConnection operations, there are two ways you can create a parser: using the graphical user interface (GUI) or the command line interface.

To create a parser using the GUI:

1. From the Actions pull-down menu on the Tasks window, select **Parsers → Create**.
2. On the Create Parser window, type the requested information.

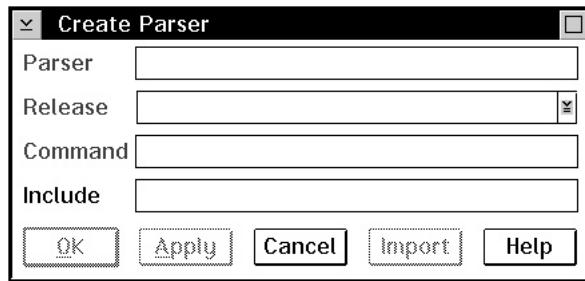


Figure 50. Create Parser window

From a command line, type the parser -create command and press Enter. The complete command syntax looks like the following:

```
teamc parser -create name -command name -release name -family name
               [-include paths]
               [-become user_name] [-verbose]
```

No matter which way you create a parser, you must specify a number of attributes for it. Together with the contents of the parser command file, the following attributes define how a parser determines the dependencies for a TeamConnection part.

#### Parser

The name of the parser must be unique within a release. It can be anything you want, but for best results, establish and follow a meaningful naming convention. An example of a parser name is `c_parser`.

#### Release

This is the name of the release that contains the parser. Parsers are release-specific objects. They are not versioned within a release; therefore you can have only one version of a parser at any time in a release.

To use the parser from a previous release, you can link to a part that uses it in that release. This action copies the parser to the new release. Otherwise, you must create the parser again in the new release.

#### Command

This is the name of the command file that the parser invokes to determine the dependencies. It can be any file name that exists in the execution path of the family server at the time a build is performed. TeamConnection runs the command as a subprocess on the machine where the build processor is located.

The task of the command file is to inspect the source file and return a list of dependencies. The syntax for invoking this command is discussed in “Writing a parser command file” on page 305.

#### Include

This is a concatenated set of paths that define where the parser looks for parts when processing the set of dependencies returned from the command file. These dependencies come in two types:

- A dependency in which the file is stored in the TeamConnection database. For example, `hello.c` includes `hello.h`, and both files are stored in the TeamConnection database. During a build, these dependencies must be extracted to a path accessible by the build processor. Because a build extracts parts from TeamConnection, anyone requesting a build needs to have PartExtract authority to all parts involved in the build.

- A dependency on a file that is not stored in the TeamConnection database. An example of such a dependency is `stdio.h`, which is typically stored in a compiler's include path and not in the TeamConnection database.

Each path named in **Include** is queried in the TeamConnection database to see if it contains a part matching the dependency name. For example, suppose you define a parser named `c_parser` with an include path as follows:

```
src\include;src\package;.;src\comm\include;
```

One of the parts to which this parser is attached, `src\example.cpp`, contains the statement `#include "example.hpp"`. Thus the command file for `c_parser` reports `example.hpp` as a dependency of `src\example.cpp`. The parser concatenates each path listed in `c_parser`'s include path with the name `example.hpp`, then inspects the contents of the TeamConnection database to see if a part with that name exists. So the TeamConnection database is queried first to find `src\include\example.hpp`, then `src\package\example.hpp`.

The period (.) in the include path tells TeamConnection to concatenate the path of the part to which the file is a dependent with the dependent's file name. In this example, that means the TeamConnection database is queried to find a part named `src\example.hpp`.

---

## Putting a parser to work

For an application to use a parser, the parser must be attached to the TeamConnection parts that it will check for dependencies. Unlike a builder, a parser is attached to the *input* part rather than the output.

To attach a parser to a part, do one of the following:

- From the GUI, select **Parts** → **Modify** → **Properties** from the Actions menu of the TeamConnection Tasks window. On the Modify Part Properties window, type the name of the parser.

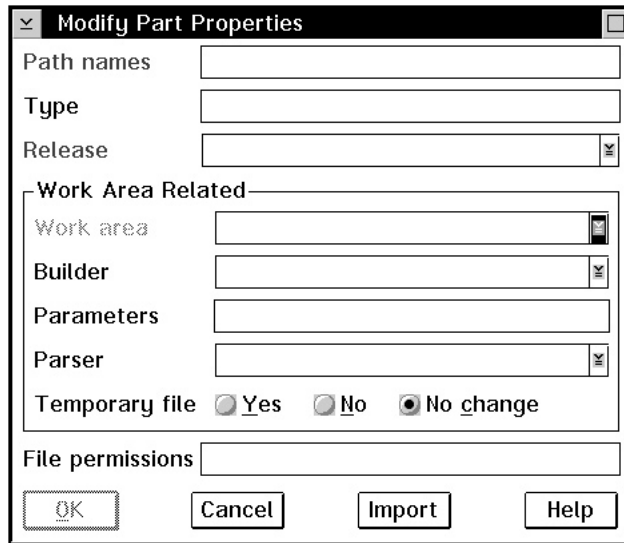


Figure 51. Modify Part Properties window

- At a command prompt, type the following and press Enter:

```
teamc part -modify part -parser name -release name
          -family name
```

The complete syntax for this command is described in the *Commands Reference*

You can also attach a parser to a part when the part is created.

After you attach a parser to a part, it is ready for action. The next time the part is used in a build, the parser will invoke its command file, which will report back a list of dependencies.

Using a parser does not keep you from defining dependencies manually by using the GUI or the part -connect command. If you explicitly define a dependency in this way, the dependency is not deleted unless you delete it, regardless of whether the parser would recognize it as such.

---

## Removing a parser from a part

If you no longer want to use a parser to determine dependencies for a part, do one of the following:

- From the GUI, select **Parts** → **Modify** → **Properties** from the Actions menu of the TeamConnection Tasks window. On the Modify Part Properties window, type null in the **Parser** field.



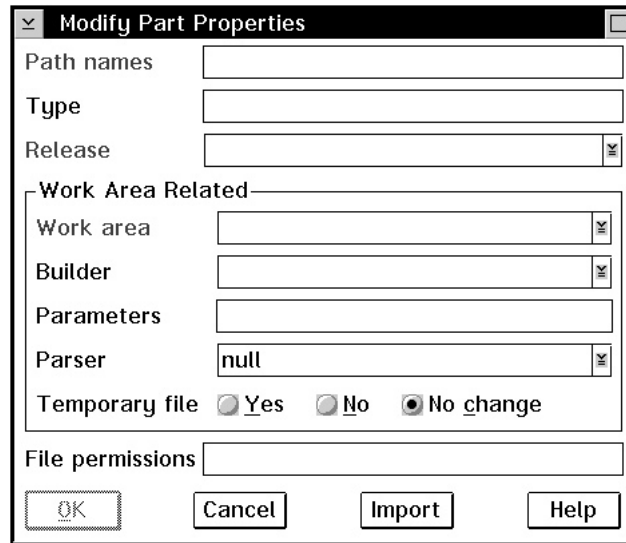


Figure 52. Modify Part Properties window

- From a command line, type the following:  

```
teamc part -modify name -parser null
          -release name -family name
```

## Writing a parser command file

A parser command file accepts two parameters as input:

- *source file*—the name of the file that contains the source to be parsed.
- *dependency list file*—the name of a file into which the names of the dependent files should be written, one per line. For example, the contents of the file might look like this:

```
hello.h
stdio.h
```

Both the source file and the dependency list file are created by the TeamConnection family server. They are erased after the parse is complete.

To write a command file, write a program, in any language, that does the following:

1. Reads the source file
2. Determines which other files are used by it
3. Writes out the list of such files into the dependency list file

For example, for a C source file, the program could report a list of all the files included by the source file (using `#include` statements). For a COBOL program, `COPY` statements would be the cue. TeamConnection ships a sample of a command file named `fhbopars.cmd`. It is written in REXX.



## Chapter 24. Building an application: an example

This chapter uses an extended example to describe in more detail how each of the components of the build function work together. This example walks through the control flow for a sample application, explaining what happens at each step.

These are the tasks involved in building our sample application, msgcat.exe:

Task	Page
Starting build processors and build agents	308
Creating builders and parsers	309
Creating the application build tree	309
Starting the build	313
Monitoring the build	318
Building in spite of errors	319
Forcing a build of all parts	319
Finding out which parts will be built	320
Canceling a build	320

We will use a simple example build tree that looks like the following:

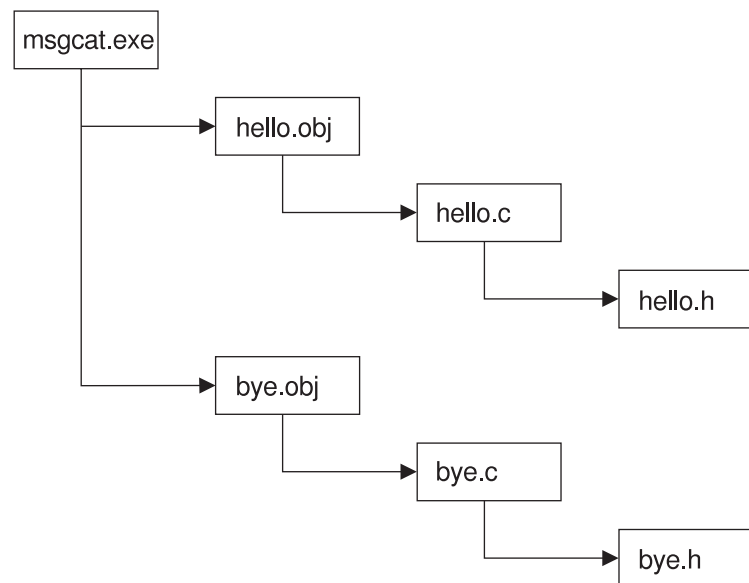


Figure 53. Sample build tree

For more examples of build trees, see “More sample build trees” on page 321.

In terms of the build object model, the objects that make up this tree look like this:

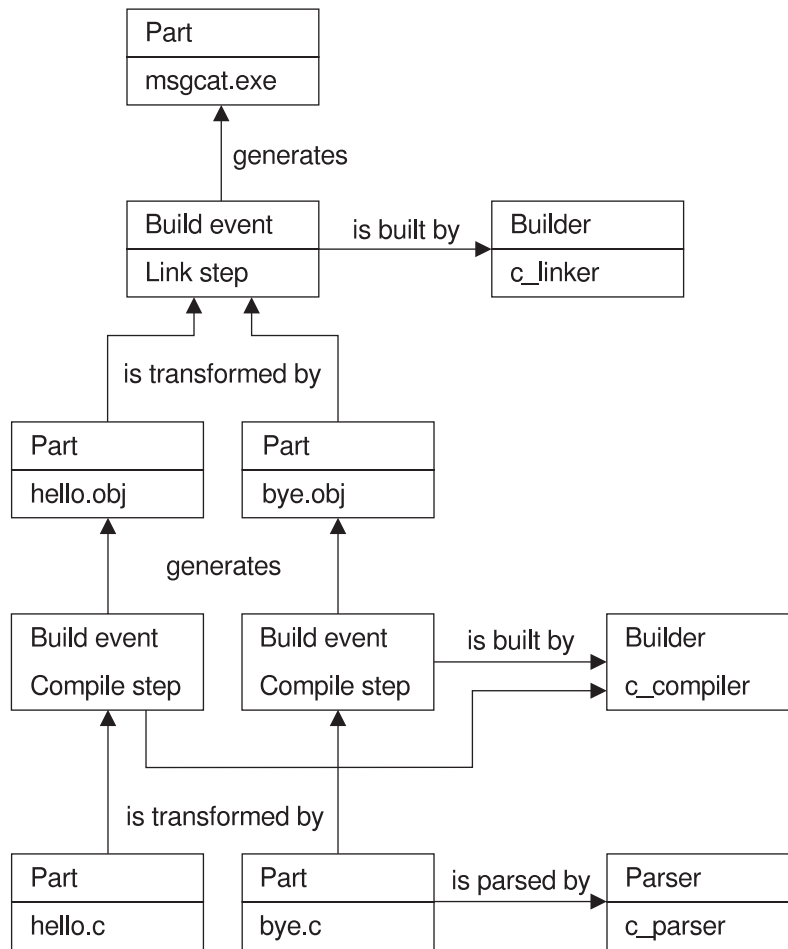


Figure 54. Sample build object model for msgcat.exe

## Starting the build processors and build agents

The software development team in our example is building large applications using a family named testfam, so they set TC\_FAMILY to testfam. They plan to spread the work across several build processors, taking advantage of TeamConnection's ability to perform multiple build events simultaneously.

Mark, the build administrator, has installed a number of build processors and build agents on the team's machines, for building OS/2 and MVS applications. As he starts them (in pairs), he groups the build agents into *pools*, according to the work he expects to use them for.

Mark plans for the following pools:

**mvs** For MVS builds

**pool1** For normal OS/2 builds

**pool2** For fast, high-priority OS/2 builds

Each pool is formed as Mark starts build agents and assigns them to it. Among the build processor-build agent pairs that he starts are the following:

```
teamproc -s bldsock2
teamagnt -s bldsock2 -p pool1 -e os2
```

The first command starts the build processor at TCP/IP address 0705. The second command starts the matching build agent at TCP/IP address 0705. The other parameters specify the following:

- p**      The agent is assigned to the pool named pool1.
- e**      The environment is os2.

Use the teamproc and teamagnt commands to start the build processor and agent when the family server has already been started. To start the family server along with the build processor and build agent, you can use the teamcd command.

---

## Creating builders and parsers

For the parts of the application that are written in C language, Mark creates the following:

- A builder named c\_compiler, to do the compiles
- A builder named c\_linker, to do the links
- A parser named c\_parser, to check for dependencies

For both builders Mark specifies os2 as the **Environment**, the same as that of the build agent started earlier at TCP/IP address 0705. Build events that use these builders (c\_compiler and c\_linker) can take place on this build agent-build processor pair.

After he creates the builders and parsers for the applications, Mark spreads the following information to the programmers who will be using them:

- The names of the build pools
- The names and purposes of the builders and parsers

---

## Creating the build tree for the application

At this point, Greg begins defining the build tree for his portion of the application, as shown in Figure 53 on page 307. He has already created the files hello.c, hello.h, bye.c, and bye.h in the TeamConnection database. Now he does the following:

1. Creates a place-holder part for the output of the link step. This file, msgcat.exe, is the target for the entire build, the output of linking hello.obj and bye.obj using the builder c\_linker, and the parent of hello.obj and bye.obj. Because the file has no contents initially, he selects **No source** (or specifies -empty on the command line), to identify it as a place holder.

Using the GUI, he can create this file by selecting **Create** from the **Actions** → **Parts** menu of the Tasks window, and completing the fields as shown in the following illustration:

Figure 55. Create Parts window

Using the command-line interface, he can create the part by issuing the following command:

```
teamc part -create msgcat.exe -builder c_linker -binary -empty
-release 9503 -workarea 223 -component comp1
```

2. Creates two place-holder parts for the output of compiling the .c files. These parts are the output of the compile step; c\_compiler, the builder that manages that step, is attached to both of them. He indicates that they are input to their parent file, msgcat.exe.

Using the GUI, he can create these files by selecting **Create** from the **Actions** → **Parts** menu of the Tasks window, and completing the fields as shown in the following illustration:

Figure 56. Create Parts window

Using the command-line interface, he can create the parts by issuing the following command:

```
teamc part -create hello.obj bye.obj -builder c_compiler -binary -empty
-release 9503 -workarea 223 -component comp1 -parent msgcat.exe -input
```

3. Attaches the parser c\_parser to the .c files.

Using the GUI, he can attach the parser to these files by selecting **Modify → Properties** from the **Actions → Parts** menu of the Tasks window, and completing the fields as shown in the following illustration:

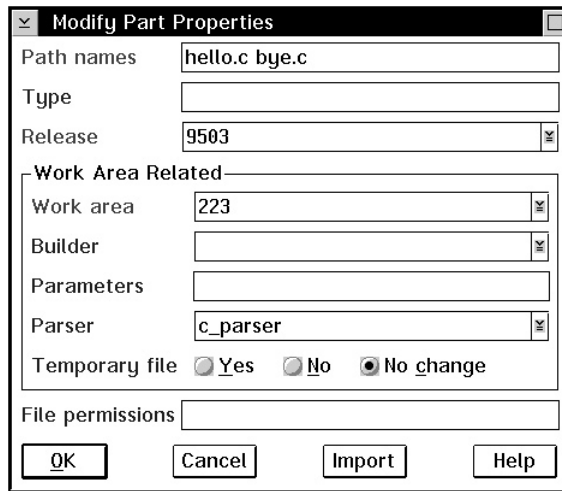


Figure 57. Modify Part Properties window

Using the command-line interface, he can attach the parser to these parts by issuing the following command:

```
teamc part -modify hello.c bye.c -parser c_parser -release 9503
-workarea 223
```

Remember, the parser is attached to an *input* file.

4. Connects the .c files into the build tree.

Using the GUI, he can connect these files by selecting **Connect** from the **Actions** → **Parts** menu of the Tasks window, and completing the fields as shown in the following illustration. He needs to execute this function twice: once to connect hello.c to hello.obj and once to connect bye.c to bye.obj.

Using the command-line interface, he can connect these parts by issuing the

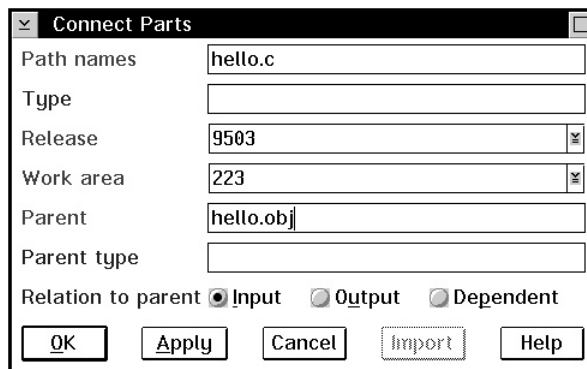


Figure 58. Connect Parts window

following commands:

```
teamc part -connect hello.c -parent hello.obj -input -release 9503
-workarea 223
```

```
teamc part -connect bye.c -parent bye.obj -input -release 9503
-workarea 223
```



- Now, Greg can see the build tree in the GUI. From the **Objects** pull-down menu on the Tasks window, he selects **Parts** → **View build tree**. The BuildView Filter window is displayed; from here he can bring up the build tree.

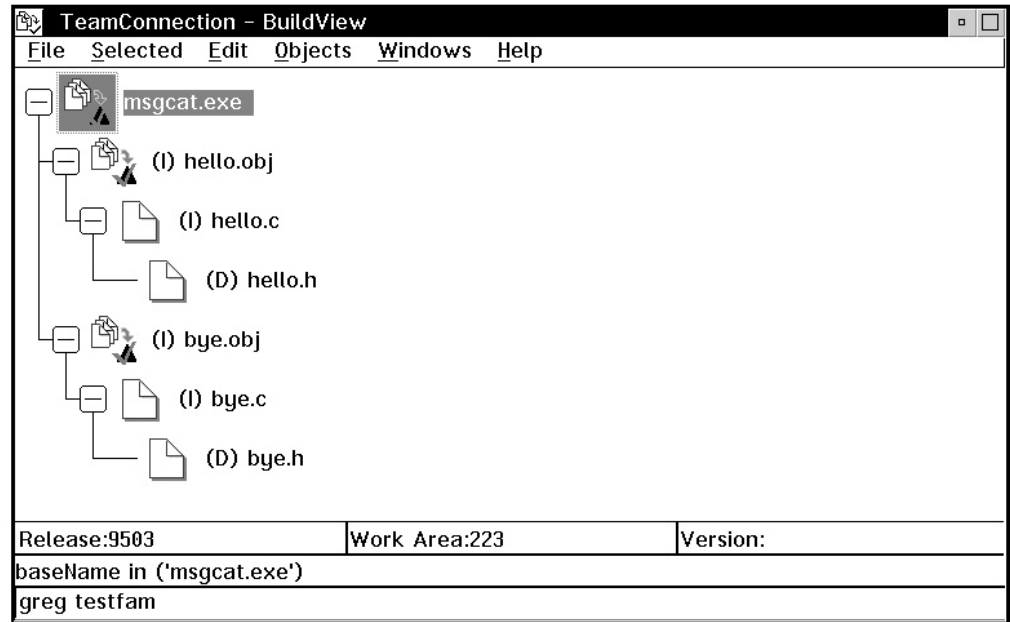


Figure 59. The build tree display

## Starting the build on the client

After much hard work on his source code, Greg is ready to start building his application.

Using the GUI, he can start the build by selecting **build** from the **Actions** → **Parts** menu of the Tasks window, and completing the fields as shown in the following illustration:

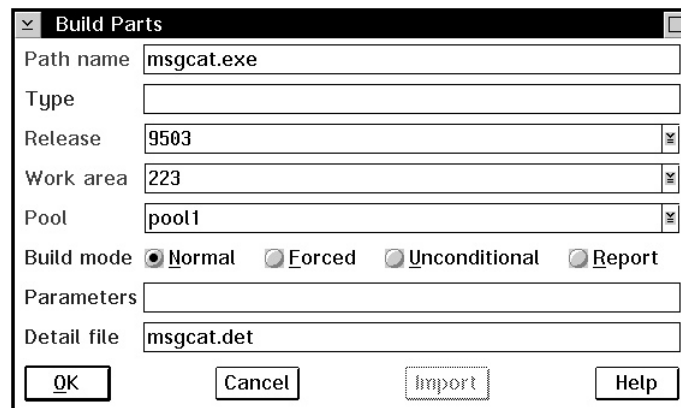


Figure 60. Build Parts window

Using the command-line interface, he can start the build by issuing the following command:

```
teamc part -build msgcat.exe -release 9503 -workarea 223  
-pool pool1 -normal -detail msgcat.det
```

This command specifies the following:

#### **Build target**

The name of the part at the top of the build tree, msgcat.exe, which is the final output of this build. TeamConnection uses the build target to determine the scope of the build.

#### **Work area**

The version of the TeamConnection parts and build tree to be used when performing this build. This version is completely specified by naming the family, release, and work area: in this case, -release 9503 -workarea 223. The output of the build is placed in this work area.

#### **Build pool**

The set of build agents that should be used to process the build request, as defined when the build agents are started. The pool pool1 includes the build agent started in “Starting the build processors and build agents” on page 308 .

#### **Build mode**

How the build takes place. Possible values for this build option include the following:

##### **Normal**

Builds only the parts that are out-of-date. Processing stops after the first error is returned.

**Force** Builds all parts, even if they are not out-of-date. Processing stops after the first error is returned.

##### **Unconditional**

Builds only parts that are out-of-date but continues processing even if errors are returned. Note that outputs are not rebuilt for inputs that have failed.

##### **Report**

Gives a preview of what would be built if you invoked a build. The report identifies what steps would occur without any translations taking place.

In our example, Greg specifies -normal, which is the default. In this mode, only the parts that are stale with respect to their inputs are rebuilt. In other words, only the minimum amount of work to bring everything up-to-date is performed. “Determining the build scope” on page 315, gives more information about how TeamConnection determines which parts to build. “Running a build in spite of errors” on page 319 and following sections provide examples of using the other build modes.

In normal mode, the build is halted if an error is found. Any remaining build events in the build scope are canceled, but any build events already performed are not undone.

#### **Detail file name**

The name of an output file in which TeamConnection stores the collected stdout and stderr of the build scripts.

When Greg starts the build, the information from the command is passed to the testfam family server over TCP/IP. At this point, Greg's TeamConnection client waits to receive confirmation from the family server that the build request was received and is being processed.

**Note:** Only one build is allowed in a work area at one time (though the build events that make up the build might be distributed to different build agents on a number of machines). So if Greg is sharing work area 223 with Barbara, she cannot issue a part -build command in that same work area until Greg's build is complete.

TeamConnection handles the next parts of the build process automatically.

## Determining the build scope

The next step is for the build function to determine the build scope. The build scope is a set of all the build events that need to be done to bring a particular build target up-to-date with respect to its inputs. When a part is built, TeamConnection marks both the input and output with the time of the build; when a later build takes place, the output part is rebuilt if its build time is different from the input part's.

In our example, the TeamConnection family server looks at the build tree for msgcat.exe. If the build time on an input part is different than the build time on its output, then the output part must be rebuilt. (Remember that the output of one build event can also be the input to the next.)

In our example, suppose the update times look like this.

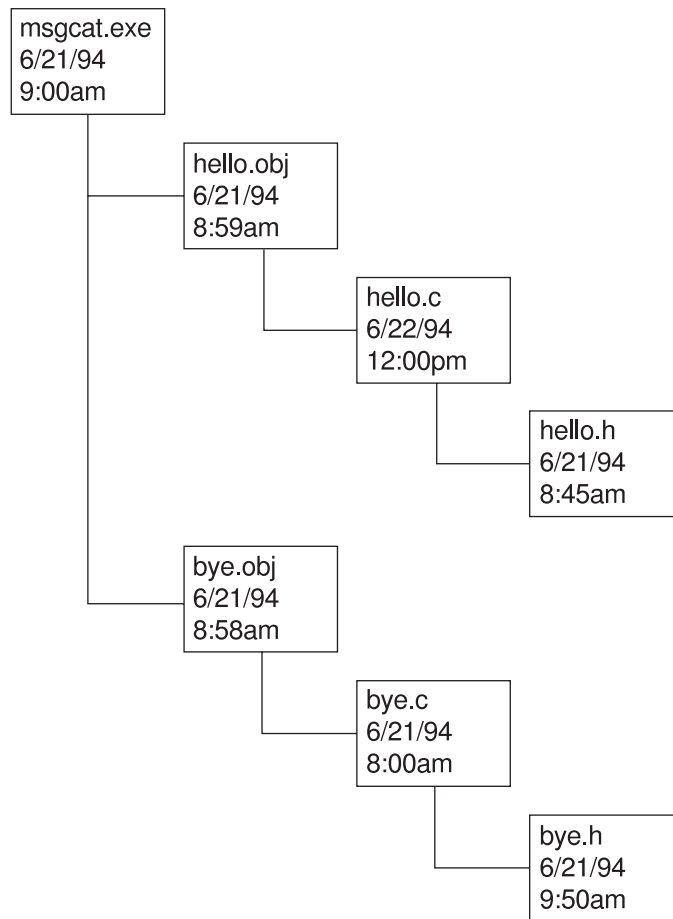


Figure 61. Build tree showing build times

In this case, Greg's build command for msgcat.exe causes the creation of a build scope including the following build events:

- A build event for creating hello.obj—that is, a compile of hello.c
- A build event for creating msgcat.exe—that is, a link of hello.obj and bye.obj

The build event for creating bye.obj is not included in the build scope, because bye.obj is up-to-date with respect to its input bye.c. The build time on bye.obj is not equal to that of its input, bye.c.

Greg wants to make sure that bye.obj is rebuilt, so he *touches* bye.c or bye.obj, using the TeamConnection command part -touch. This command marks the part as out-of-date in the work area so that it is included in the next build. Now when he starts a build of msgcat.exe, the build scope includes the build event to create bye.obj.

The resulting build scope contains three build events: two compiles and a link. Notice that the two compiles can be performed in parallel, for faster performance.

At this point any parsers associated with parts in the scope of this build are invoked. The information returned from the parsers is used to analyze dependencies.

TeamConnection can determine if a part has already been built in another context using identical inputs and time stamps. Suppose, for example, a part is build in work area 913 and that work area has been integrated. Next, a new work area, 914, is created in the same release and the same part is requested to be built in that work area. Since the time stamps in work area 913 and the release are the same as in work area 914, the build does not have to execute and the build outputs are copied to work area 914.

---

## Adding the job to the job queue

The next step is for TeamConnection to add the build scope to the testfam family's job queue. This step ensures that the build events in it are picked up and processed by any available build agents. After adding the build scope to the job queue, the family server gives Greg's client confirmation that the build is being processed. It also reports some statistics about the build, such as the number of build events. The client reports this data to Greg; it then waits to receive and report the outcome of each build event as it happens.

---

## Picking up the work orders

Let's look at what has been happening while the preceding steps have been going on. Each of the build agents that Mark started earlier is actively polling the family's job queue to see if there is any work they can perform.

Each build agent looks at the top-most build scope in its job queue. If it contains a build event that this build agent can perform (that is, one for the pool and environment it was started for), the build agent takes the event and starts to process it. If it does not find a build event it can process, it waits awhile and then tries again.

Suppose that Mark started four build processors and their corresponding build agents, two in pool1 and two in pool2. That means that, as soon as TeamConnection determines the build scope for msgcat.exe, each of the build agents in pool1 finds a build event it can perform, one for compiling hello.c and one for compiling bye.c.

---

## Putting the build processors to work

Each build agent that finds an event it can perform sends a description of the event to its corresponding build processor over TCP/IP. It then sends to and receives from the TeamConnection database any parts data the processor needs.

In our example, each of the two build agents in pool1 sends a description of the compile event to its connected processor and then waits to answer any requests from the processor for parts data.

---

## Putting the build scripts to work

At this point, the build processor looks at the description of the event it has been asked to perform, then checks its cache for each part and the build script it needs. If it does not find them there or if the cached parts are out of date, it asks the build agent.

It then invokes the build script, passing it the names of the input and output parts and the parameters specified on the builder. The parts created by the build script and the return code generated by it are sent back to the waiting build agent. The build agent then updates the contents of the TeamConnection database.

In our example, each of the two build processors receives a compile event to perform. Each asks its corresponding build agent to extract the .c source files it needs from the TeamConnection database and the contents of the build script for the c\_compiler builder. It then runs the build script.

The results (the .obj files and the return code) are sent back to the build agents. After updating the TeamConnection database, the build agents re-enter their polling loop to see if any more build events await their attention.

Because the compile steps are performed in parallel, Greg can build this application a little more quickly than if they had happened in serial mode. In this simple example, the difference is hardly noticeable; but in a large build of hundreds of parts, with multiple build processors available on a local area network, the performance improvement can be enormous.

---

## Finishing the job and reporting the results to the user

The processing described by the previous two steps is repeated until there are no more build events that comprise the build scope. The results of the build are displayed in the Build Progress window or in stdout. At this point the build is complete.

To complete our example, the previous two steps are repeated to complete the link step, using either of the two build agents in pool1. Greg now can extract the resulting executable from TeamConnection, using the part -extract msgcat.exe command, and run it.

---

## Monitoring the progress of a build

During the course of a build, you can monitor its progress in several ways:

- If the build was started from the command line, by issuing the report -view partview command against the work area in which you are building. From this report, you can determine the states of the parts. Use the part -viewmsg command to see the build messages issued because of a failed build. For complete syntax of these commands, refer to the *Commands Reference*
- If the build was started from the GUI, in the Build Progress window. You can find the same information by looking at stdout.

Greg can see how the build is progressing by checking the Build Progress window. For example, he might see these messages:

```
6021-301 Invoking Parser c_parser for hello.c
6021-303 A successful parse resulted from using the parser c_parser. The
        parser return code is 0
6021-301 Invoking Parser c_parser for bye.c
6021-303 A successful parse resulted from using the parser c_parser. The
        parser return code is 0
6021-700 Number of distinct build events for this build: 3.
Build of 'hello.obj' started at '15:33:47 1995-08-10'
via a build agent on the host 'OCTOFVT'.
Build of 'hello.obj' successfully completed at '15:34:45 1995-08-10'.
Completed Jobs: 1
```

```
Remaining Jobs: 2
Build of 'bye.obj' started at '15:34:49 1995-08-10'
via a build agent on the host 'OCTOFVT'.
Build of 'bye.obj' successfully completed at '15:35:22 1995-08-10'.
Completed Jobs: 2
Remaining Jobs: 1
Build of 'msgcat.exe' started at '15:35:26 1995-08-10'
via a build agent on the host 'OCTOFVT'.
Build of 'msgcat.exe' successfully completed at '15:35:56 1995-08-10'.
Completed Jobs: 3
Remaining Jobs: 0
Processing Completed for 'msgcat.exe'.
```

To see the commands that TeamConnection issued during the build, he can look at the detail file that he specified in the part -build command.

---

## Running a build in spite of errors

If you find that a build is stopping because of errors, you can check the build detail file or the Build Progress window for the cause. If the error is minor, you might decide to run the build despite the errors—for example, when you are debugging. To do this, specify that you want the build to complete unconditionally.

In our example, when Greg builds msgcat.exe for the first time, he wants to find and correct any errors that occur during the build, so he uses the following command:

```
teamc part -build msgcat.exe -release 9503 -workarea 223 -unconditional
```

As in normal mode, only the parts that are stale with respect to their inputs are rebuilt; only the minimum amount of work to bring everything up-to-date is performed.

However, even if an error is found, the build continues if possible. As with normal mode, if the build is halted, any build events remaining in the build scope are canceled. Any build events already performed are not undone.

---

## Building all parts, regardless of build times

To make sure that **all** parts in the build tree get built, whether or not they are stale, you specify the -force parameter on the part -build command.

In this mode, all parts that are descendants of the build target are rebuilt, no matter what.

In our example, Greg can force TeamConnection to build all parts in the msgcat.exe build tree using the following command:

```
teamc part -build msgcat.exe -release 9503 -workarea 223 -force -pool pool1
```

If an error occurs, the build is halted. Any remaining build events in the build scope are canceled, but any build events already performed are not undone.

---

## Finding out which parts will be built

Before running a build of a large application, you might want to find out exactly which parts will be built. If you specify that you want to run in report mode, TeamConnection determines what will be built in a normal build and produces a report showing the results.

If Greg really wants to see which parts of msgcat.exe will be built before he runs the actual build, he can issue the following command:

```
teamc part -build msgcat.exe -release 9503 -workarea d410 -report -pool pool1
```

He sees the following report:

```
6021-301 Invoking Parser c_parser for hello.c
6021-303 A successful parse resulted from using the parser c_parser. The
        parser return code is 0
6021-301 Invoking Parser c_parser for bye.c
6021-303 A successful parse resulted from using the parser c_parser. The
        parser return code is 0
6021-700 Number of distinct build events for this build: 3.
6021-407 The builder c_compiler will be invoked.
6021-406 The builder parameters consist of:
        command:  compC.cmd
        input:    hello.c
        output:   hello.obj
        dependent: hello.h
6021-407 The builder c_compiler will be invoked.
6021-406 The builder parameters consist of:
        command:  compC.cmd
        input:    bye.c
        output:   bye.obj
        dependent: bye.h
6021-407 The builder c_linker will be invoked.
6021-406 The builder parameters consist of:
        command:  linkC.cmd
        input:    hello.obj bye.obj
        output:   msgcat.exe
        dependent:
```

The report shows that bye.obj and msgcat.exe must be rebuilt.

---

## Canceling a build

To cancel a build that is in progress, do one of the following:

- If the build was started from the GUI, on the Build Progress window select the **Cancel Build** push button.
- If the build was started from the command line, type the following command and press Enter:

```
teamc part -build name -cancel
```

Where *name* is the part that you are building. Be sure to specify the same part name that you specified when starting the build, rather than a part that is lower in the build tree.

This command stops any further build events being performed for that build scope. Any build events already performed for that build are not undone.



For example, if Greg cancels the build of msgcat.exe when the compile steps have been completed, then the link step is not performed. However, the newly compiled hello.obj and bye.obj are left in the database, with their build times updated.

### 3.1

You cannot use the **Stop Build** button on the Build Progress window to cancel a build in progress in the Windows environment. Always use the part `-build -cancel` command instead.

---

## More sample build trees

The msgcat.exe example is just one possible build tree. Here are some others.

### Defining multiple outputs from a single build event

Figure 62 shows part of the build tree for robot.dll:

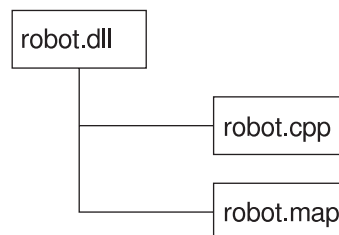


Figure 62. The build tree for robot.dll

Because the build tree shows the relationships between parts hierarchically, robot.map is a child of robot.dll, even though it is actually built from the same input part, robot.cpp. But robot.map is defined as an *output* of robot.dll. Here are the commands to set up this relationship.

First come the commands to create the parts:

```
teamc part -create robot.dll -builder dll_builder -binary -empty  
-release 9503 -component robot
```

```
teamc part -create robot.cpp -release 9503 -component robot
```

```
teamc part -create robot.map -builder dll_builder -binary -empty  
-release 9503 -component robot
```

Next are the commands to connect the parts into the build tree:

```
teamc part -connect robot.cpp -parent robot.dll -input -release 9503
```

```
teamc part -connect robot.map -parent robot.dll -output -release 9503
```

You might use this command to start the build:

```
teamc part -build robot.dll -workarea 915 -release 9503
```

The output of this build would be both robot.dll and robot.map. Any parameters specified in the teamc part -build robot.dll command would also apply to the build of robot.map.

## Synchronizing the build of unrelated parts

An entire application can require multiple separate builds. For example, in the robot application, there might be one build to create the .dll parts, another to create the .exe parts, and so on. To ensure that the entire application gets built together, you can create a part that acts as a collector, with the .dll and .exe parts as input to it.

For example, Tim creates this build tree for the robot application:

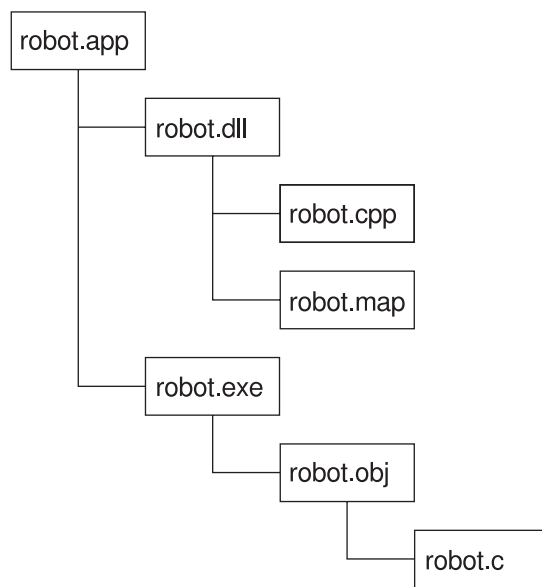


Figure 63. The build tree for robot.app

Assuming he already has the build trees for robot.dll and robot.exe set up, here is how he sets up the collector part:

1. He creates a null builder with no contents:

```
teamc builder -create nullBuilder -script null -none -environment os2  
-condition == -value 0
```

2. He creates the collector part:

```
teamc part -create robot.app -builder nullBuilder -none -release 9503  
-component robot
```

The -none flag identifies this as a part that will never have any contents.

3. Tim connects the other parts to the collector:

```
teamc part -connect robot.dll robot.exe -parent robot.app -input  
-release 9503
```

When Tim builds robot.app, the result is a build of both robot.dll and robot.exe.

---

## Part 6. Using TeamConnection to package products

<b>Chapter 25. Using TeamConnection to package a product</b>	325
Setting up your build tree for packaging	326
Setting up a build tree for the gather tool	326
Setting up a build tree for the NVBridge tool	328
Setting up a build tree for other distribution tools	329
 <b>Chapter 26. Using the Gather tool</b>	 331
Using the teamcpak command for the Gather tool	332
Command line flags	332
Examples of the teamcpak gather command	333
Writing a package file for the Gather tool	334
Syntax rules for a Gather package file	334
Keywords for a Gather package file	335
Using exit keywords in the DATA clause	337
Using exit keywords in the RULE clause	338
Using exit keywords: an example	338
 <b>Chapter 27. Using the NVBridge tool</b>	 339
Using the teamcpak command for NVBridge	340
Command line flags	341
Examples of the teamcpak nvbridge command	342
Writing a package file for NVBridge	342
Syntax rules for an NVBridge package file	343
Keywords for an NVBridge package file	343
Problem determination for NVBridge	351
NVBridge utilities	352
FHPSTAT	353
Syntax	353
Return Codes	353
Example	353
FHPOBDEL	353
Syntax	353
Return Codes	353
Example	353
FHPOBMON	353
Syntax	354
Return Codes	354
Example	354
FHPOBDIF	354
Syntax	355
Return Codes	355
Example	355
FHPISCAT	355
Syntax	355
Return Codes	355
Example	355
FHPICAT	355
Syntax	356
Return Codes	356
Example	356
FHPUCAT	356
Syntax	356
Return Codes	356

Example . . . . .	357
FHPMCAT . . . . .	357
Syntax . . . . .	357
Return Codes . . . . .	357
Example . . . . .	357
FHPVERIF . . . . .	357
Syntax . . . . .	357
Return Codes . . . . .	358
Example . . . . .	358
FHPRQPUR . . . . .	358
Syntax . . . . .	358
Return Codes . . . . .	358
Example . . . . .	358
FHPRQMON . . . . .	358
Syntax . . . . .	359
Return Codes . . . . .	359
Example . . . . .	359
FHPTRVER. . . . .	359
Syntax . . . . .	359
Return Codes . . . . .	359
Example . . . . .	360
FHPTRPUR . . . . .	360
Syntax . . . . .	360
Return Codes . . . . .	360
Example . . . . .	360

<b>Chapter 28. Using the Tivoli Software Distribution packaging tool . . . . .</b>	<b>361</b>
Using the teamcpak command with Tivoli Software Distribution . . . . .	361
Command line flags. . . . .	362
Example of the teamcpak softdist command. . . . .	362
Writing a package file for Tivoli Software Distribution. . . . .	363
Syntax rules for a Tivoli Software Distribution package file . . . . .	363
Keywords for a Tivoli Software Distribution package file . . . . .	363
Problem determination for the Tivoli Software Distribution tool . . . . .	366
Sample package file . . . . .	366

This section describes how to use the TeamConnection packaging function, which helps you automate the packaging and distribution of your applications. This section is written for the person in your organization who is responsible for software distribution.

## Chapter 25. Using TeamConnection to package a product

After you have built an application to your satisfaction, it is time to distribute it to users. This chapter describes how you can use TeamConnection to help automate the packaging and distribution steps.

TeamConnection provides the following:

- Two electronic software distribution tools:
  - **Gather**, which moves an application's parts into a single directory in preparation for distribution.



The Gather utility is available on OS/2 and Windows NT platforms.

- **NVBridge**, a bridge tool that automates the installation and distribution of software or data using IBM NetView Distribution Manager/2 as the distribution vehicle.



The NVBridge utility is available only on OS/2.

- Two sample build scripts for connecting the Gather and NVBridge tools with TeamConnection user-defined builders.
- A set of mini-utilities that can be used to develop customized electronic software distribution solutions.

To use TeamConnection in packaging a product, you might do any of the following tasks:

Task	Page
Setting up your application's build tree for packaging	326
Using the teamcpak gather command	332
Writing a package file for the gather tool	334
Using the teamcpak nvbridge command	340
Writing a package file for the NVBridge tool	342

---

## Setting up your build tree for packaging

When TeamConnection builds an application, the application's build tree identifies the parts to be built and the tools to use in building it. Similarly, when you use TeamConnection for packaging the application, the build tree can define the parts to be packaged and the tools to do it.

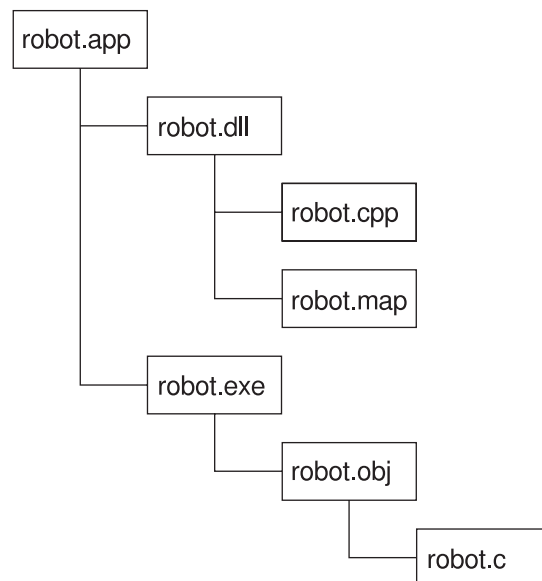
The output of a packaging step might be any of the following:

- The application's parts gathered into a new directory structure
- The distribution of the application using NVBridge
- The distribution of the application using some other distribution tool

## Setting up a build tree for the gather tool

To gather the parts of your application into a single directory for distribution, you create an output part whose builder calls the gather tool, and you make this output part the top level of the build tree.

For example, for the robot control application, `robot.app`, the build tree might look in part like this:



*Figure 64. Part of the build tree for robot.app*

After the application is built, the programming team needs to get it to the test team. They could extract the application, but doing a simple extract would preserve the existing structure, with parts contained in directories according to their application component. A better structure might be to place all of the .dll files in one directory, all of the .exe files in another, and so on. To move the parts into this structure, the test team does a different kind of build, using the gather tool.

To make this happen, Annmarie does the following:

1. She creates the top-level part for the new build tree. The name of this part is the same as the directory in which the gathered parts are to be placed. In this example, `e:\robot` is the output file from the gather step. Annmarie uses the following command:

```
teamc part -create e:\robot -none -builder gather1 -family octo
-release 9503 -workarea 410
```

2. She writes a package file that contains instructions for the gather tool and creates this file as a TeamConnection part:

```
teamc part -create robot.pkf -text -parent e:\robot -input -family octo
-release 9503 -workarea 410
```

For more information, see “Writing a package file for the Gather tool” on page 334 .

3. She creates a builder, `gather1`, that calls the gather tool:

```
teamc builder -create gather1 -script gather.cmd
-parameters "-o -x" -release 9503
-environment os2 -condition == -value 0 -family octo
```

`gather.cmd` is a sample build script that is shipped with TeamConnection. It specifies the `teamcpak gather` command.

4. She connects `robot.exe` and `robot.dll` to `e:\robot` as inputs:

```
teamc part -connect robot.exe -parent e:\robot -family octo
-release 9503 -workarea 410
```

```
teamc part -connect robot.dll -parent e:\robot -family octo
-release 9503 -workarea 410
```

5. She also connects a readme file for the application:

```
teamc part -connect read.me -parent e:\robot -family octo
-release 9503 -workarea 410
```

As a result of Annmarie’s work, the build tree for `e:\robot` looks like this:

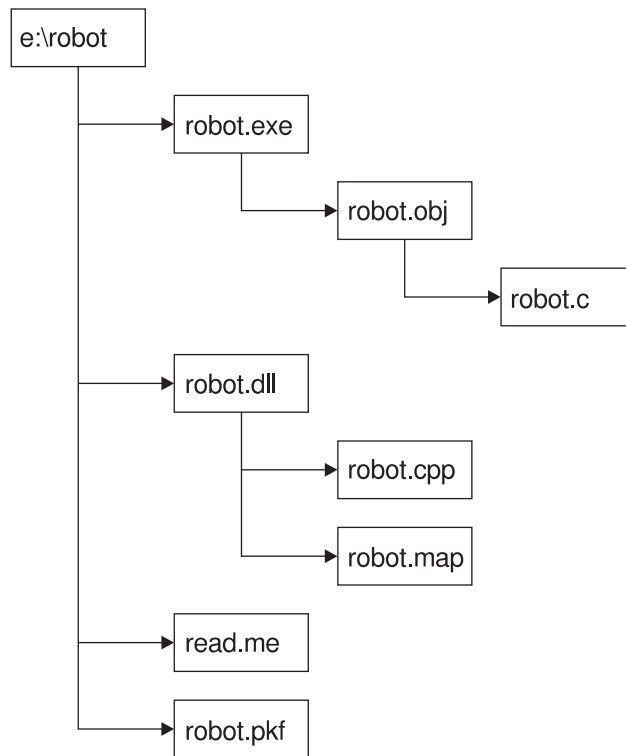


Figure 65. Adding the gather step to the build tree

The package file, robot.pkf, specifies the directories into which the robot files are gathered, with e:\robot as the target root directory. When Annmarie builds e:\robot, the .dll files are placed in e:\robot\bin; the .bin files are placed in e:\robot\bin. Instead of extracting the built application from TeamConnection, the test team can pull the application from e:\robot.

If Annmarie wants to gather the same files into a different target directory, all she needs to do is write a different package file and connect the parts to a different parent.

## Setting up a build tree for the NVBridge tool

If you have NetView DM/2 on your LAN, you can use the NVBridge tool to distribute your application to users. Setting this up is similar to setting up the build tree for the gather tool.

For example, Marylin, the packaging administrator for our robot development team, does the following:

1. She creates the top-level part for the new build tree. In this example, robotn.out is the output file from the NVBridge build. Marylin uses the following command:

```
teamc part -create robotn.out -none -builder NVB1
-release 9503 -workarea 817 -family octo
```

2. She writes a package file that contains instructions for the NVBridge tool and creates this file as a TeamConnection part:

```
teamc part -create robotnvb.pkf -text -parent robotn.out
-release 9503 -workarea 817 -family octo
```



For more about creating this package file, see “Writing a package file for NVBridge” on page 342.

3. She creates a builder, `nvb1`, that calls the NVBridge tool:

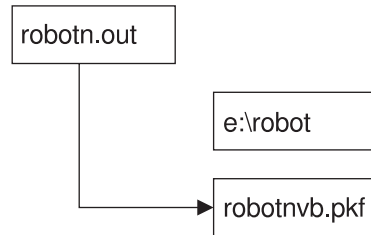
```
teamc builder -create nvb1 -script nvbridge.cmd -release 9503  
-environment os2 -condition == -value 0 -family octo
```

The build script for this builder specifies the `teamcpak` command, with its parameters.

4. She connects `e:\robot` and `robotnvb.pkf` to indicate that they are input to `robotn.out`:

```
teamc part -connect e:\robot robotnvb.pkf -parent robotn.out  
-family octo -release 9503 -workarea 817
```

As a result of Marylin’s work, the build tree for `robotnvb.out` looks like this:



*Figure 66. Adding the NVBridge step to the build tree*

When Marylin builds `robotn.out`, NVBridge uses the rules in the package file to issue NetView DM/2 commands. The result is that the entire package is distributed using NetView DM/2.

In this example, Marylin uses NVBridge to distribute the output of the gather tool. This step is not required, but the gather tool is good preparation for distributing an application to users.

## Setting up a build tree for other distribution tools

This process is similar to setting up for NVBridge. You create a top-level part in the build tree and a builder that invokes the distribution tool.



---

## Chapter 26. Using the Gather tool



The Gather tool is available on OS/2 and Windows NT platforms. The command syntax for using the tool is the same for both platforms.

The Gather tool automates the movement of software and data from one directory to another on the same machine to prepare a package for electronic distribution. It can copy or erase files; it can create or delete directories.

This tool takes a list of input files and moves them into a directory structure as directed by a package specification file. You specify the target root directory path in this file, along with a collection of rules that instruct which files to copy to which directories. How these files and directories are actually handled is controlled via option flags.

By writing different package specification files, you can take the same input files and transfer them into different target directory structures.

Take the robot application as an example. We previously showed one possible directory structure, with each subdirectory containing files with the same extension:

```
e:\robot
  \dll
    hand.dll
    optics.dll

  \exe
    hand.exe
    optics.exe
```

By writing a different package file, you might put both .dll and .exe files in the same target directory:

```
f:\robot
  \bin
    hand.dll
    optics.dll
    hand.exe
    optics.exe
```

You can build both target directories concurrently.

---

## Using the teamcpak command for the Gather tool

To start the Gather tool, use the teamcpak command. This command is found in the directory where the TeamConnection family server is installed. If it is started from a build script, it does not need to be in the execution path of the machine from which the build is started.

The complete command syntax for teamcpak gather looks like the following; you must supply a value for the words that start with a capital letter, such as String. You must specify the command parameters in the order shown.

```
teamcpak [-i] [-o "String"] gather Input_file...
```

Where

**-i** Specifies that only one *Input\_file* is specified in the command: an include file containing the list of input files. This parameter is optional.

If you specify -i, it must precede the gather flag.

**-o " String"**

Specifies that the string listed in quotes be passed to the Gather tool. The opening quote must be followed by a blank. For a list of possible flags to be passed, see "Command line flags".

This parameter is optional. If you do not specify -o, the default settings for the tool are used.

If you specify -o, it must precede the gather flag.

**gather**

Specifies the tool to be invoked. If you specify -i or -o, they must precede this value.

**Input\_files**

Specifies the files to be copied and the name of the package specification file. You can specify this parameter in these ways:

- Specify the name of an include file, whose contents is a list of input files. One of these input files must be a package specification file with the extension .pkf. In this case, you must also specify the -i parameter.
- Specify a list of two or more files. One of the files must be a package specification file with the extension .pkf.
- Specify the directory from which the files are to be copied and the name of the package specification file.

If more than one package file is listed, the first package file on the command line or in the include file is used, and the others are treated as ordinary files.

## Command line flags

You can specify the following flags in the teamcpak command, using the -o parameter. All of these flags are optional. If you do not specify a flag, the teamcpak command runs using defaults.

**-a** Assume that the target tree structure might not exist. If a required directory does not exist, create it and continue processing.

This flag cannot be specified if the -t flag is specified.

If neither `-a` nor `-t` is specified, the default is to assume that the desired tree structure already exists. No verification is performed to confirm that the directories exist. If they do not, the condition is detected while the package file rules are being processed. If you stop the `teamcpak` command, some target directories might contain updated files.

- t** Ensure that the target tree is exactly the tree specified in the package file. If a directory of the same name exists, the Gather tool does the following:
  - Erases the entire contents of the directory and all of its subdirectories
  - Destroys the directory and all subdirectories
  - Performs a `mkdir` command to create the entire tree structure again as specified in the package file

This flag cannot be specified if the `-a` flag is specified.

If an `rmdir` command fails during processing, the `teamcpak` command stops.

If neither `-a` nor `-t` is specified, the default is to assume that the desired tree structure already exists. No verification is performed to confirm that the directories exist. If they do not, the condition is detected while the package file rules are being processed. If you stop the `teamcpak` command, some target directories might contain updated files.

- m** Accept missing source files.

If this flag is not specified, the default is to ensure that at least one file matches each source specification in the package file. If a match is not found, the Gather tool stops processing.

- d** Accept duplicate files. If a file is found on the target directory that matches the source file specification, it is overwritten by the source file.

If this flag is not specified, the default is to ensure that no files on the target match the source file specification. For example, if the source specification is `g*.c`, and `greg.c` is found on the target, the Gather tool stops processing.

- c** Clean up the target directories. Erase all files on all target directories that existed before writing source files to these directories. No confirmation messages are issued, and permission errors are ignored.

If this flag is not specified, the default is to write the source files into the target directories without erasing existing files.

- e** End with delete. This action removes all source files and directories after the Gather tool successfully completes.

If this flag is not specified, the default is to end without deleting source files and directories.

- x** Abort without recovery. If the program does not end successfully, no attempt is made to restore the file system.

If this flag is not specified, the Gather tool attempts to restore the file system if the program does not end successfully. To do this, the tool first backs up the file system. The backup directory is the value of the `TMP` environment variable.

## Examples of the `teamcpak gather` command

The following are examples of the `teamcpak gather` command.

```
teamcpak gather d:\demoapp demoapp.pkf
teamcpak gather a.exe b.exe \help\*.hlp demoapp.pkf
```

In the first example, an input source directory is specified. In the second example, a list of files is specified. In both cases, the files are to be copied into target directories as specified in the demoapp.pkf file.

```
teamcpak -i -o " -t -m -x" gather myfiles.lst
```

The file myfiles.lst contains a list of files to be transformed by the Gather tool, and the name of the package file to be used in the gather. The -o "-t -m -x" parameter passes three flags to the Gather tool:

- -t specifies that, if the target directories already exist, they be destroyed and recreated.
- -m specifies that processing continues even if a source file cannot be found.
- -x specifies that, if the program does not end successfully, the file system is left as is, with no attempt to restore it.

---

## Writing a package file for the Gather tool

Use the package file to specify the target directories and the rules for copying files for a gather operation. You can also specify user exit programs to run before, during, or after the gather operation.

A sample package file named gather.pkf is shipped with TeamConnection. You can customize it for your own gather operations.

## Syntax rules for a Gather package file

Follow these syntax rules when you write a package file:

- Package files are free format. Text is not positional, and many statements can exist on the same line.
- Comments can appear anywhere within the file. Use the characters #| and |# as delimiters, as shown in the following example:  
#| This is a comment |#
- Package file keywords must be prefixed with a left parenthesis and must have a corresponding balanced right parenthesis to end the scope of the keyword.
- If the value for a keyword is a string that contains blanks or parentheses, enclose the string in double quotes.

The following shows the syntax of a package file for the Gather tool. Keywords must appear in the order shown. The first letter of an argument is capitalized; you must supply these values.

```
(DATA
  (PACKAGEFORMAT gather)
  (TARGETROOT Filename)
  (RULE
    (SOURCE Filename...)
    (TARGET Path)
    [(EXITPRIOR String... | EXITREPLACE String... | EXITPOST String...)] )
  )
  .
  .
  [(EXITPRIOR String...)]
  [(EXITPOST String...)]
)
```

## Keywords for a Gather package file

**DATA** This keyword is required. It must be the first keyword in the package file, and it can be specified only once.

All other keywords are nested within the DATA clause.

### **PACKAGEFORMAT gather**

This keyword is required. It can be specified only once. It tells the teamcpak command that this package file is for Gather.

### **TARGETROOT target\_root\_path**

This keyword is required. It can be specified only once.

Use this keyword to identify the target root directory. Source files are copied to this directory as specified by the RULE statements.

Follow these guidelines when you select your TARGETROOT values:

- Include the drive letter along with the target directory.
- Specify a directory that contains few if any subdirectories that are unrelated to the data you are moving.
- If you specify a drive's root directory (*drive:\*), run the teamcpak command using the defaults or only the -x or -x -a flags.
- Do not set the value of TARGETROOT to *drive:\* under the following circumstances:
  - The TARGETROOT drive is the same as the drive from which the teamcpak command is run, and you have recovery set (that is, you have not specified -o "-x").
  - The logical drive for the TARGETROOT has less than 50% free space, and you have recovery set (that is, you have not specified -o "-x").

**RULE** This keyword is required. You can use one or more RULE keywords within a Gather package file.

Each RULE clause represents a set of Gather operations targeted for one target subdirectory. A RULE clause must contain one SOURCE and one TARGET keyword. The files in the SOURCE directory are copied to the TARGET path. The target path is derived by concatenating the value of TARGETROOT with a backslash (\), followed by the value of the TARGET keyword specified in the RULE clause.

A RULE clause can also contain one user exit clause: EXITPRIOR, EXITPOST, or EXITREPLACE. For a description of the exit keywords, go to page 337.

The following example copies all \*.exe, \*.cmd, and \*.hlp files to target directory f:\demoapp\bin.

```
(DATA
.
.
(TARGETROOT f:\demoapp )
.
.
(RULE
(SOURCE *.exe *.cmd *.hlp)
(TARGET bin )
```

```
)
:
.)
```

### **SOURCE <list of file specifications>**

This keyword is required once for each RULE clause. It must be the first keyword within the RULE clause.

This keyword specifies the files to be copied to the path specified by the TARGET keyword. Specify a list of file specifications separated by blanks. You can use the wildcard characters supported by OS/2 or Windows NT.

The directory from which these files are copied depends on how the input files are specified in the teamcpak command:

- If the teamcpak command specifies a source directory, the files specified in the SOURCE keyword come from that directory or subdirectories of it. The full path of the source files is constructed by concatenating the directory from the teamcpak command with a backslash (\), followed by the file specifications found in the SOURCE keyword. You can specify subdirectories in the SOURCE file specifications.
- If the teamcpak command specifies a list of files, these files are first copied to a temporary directory, then copied from there to the TARGET directories. In this case, you can use OS/2 or Windows NT wildcards to specify multiple file names in the SOURCE file specifications, but you cannot specify subdirectories.

In the following example, directory d:\demoapp is specified on the teamcpak command:

```
teamcpak -o "-x -t -m" gather d:\demoapp demoga.pkf
```

The resulting source path is the concatenation of d:\demoapp with the SOURCE file specifications. Therefore, all of the .exe files in the directory d:\demoapp\bin are copied to the target directory e:\demoapp\bin.

```
(DATA
  (TARGETROOT e:\demoapp)
  .
  .
  (RULE
    (SOURCE bin\*.exe)
    (TARGET bin)
  )
  .
  .
)
```

In the following example, a list of input files is specified on the teamcpak command:

```
teamcpak -o "-x -m" gather c:\a.exe c:\b.exe d:\rexx\*.cmd demoga.pkf
```

The resulting source path for the files in the SOURCE clause is the concatenation of the teamcpak temporary directory with the SOURCE file specifications. Therefore, the source for the \*.exe files is d:\teamcpak.@@@\*.exe. The input files d:\teamcpak.@@@a.exe and d:\teamcpak.@@@b.exe are copied to the directory e:\demoapp.

```
(DATA
  (TARGETROOT e:\demoapp)
  .
  .
```



```

(RULE
  (SOURCE *.exe )
  (TARGET targetroot)
)
:
.
)

```

### **TARGET Target\_path**

This keyword is required one time in each RULE clause. It must follow the SOURCE keyword.

The value specified by this keyword is used to construct the target path into which the files specified by the SOURCE keyword are copied. The value of the TARGETROOT keyword is concatenated with a backslash (\), followed by the value of the TARGET keyword.

If you specify targetroot as the value, files are copied directly to the target root directory, not to a subdirectory.

In the first RULE clause of this example, files are copied to the target directory f:\demoapp\bin\files. In the second RULE clause, the target directory is f:\demoapp.

```

(DATA
  (TARGETROOT f:\demoapp )
  :
  .
  (RULE
    (SOURCE *.bin *.dll )
    (TARGET bin\files )
  )
  (RULE
    (SOURCE *.hlp )
    (TARGET targetroot )
  )
  :
  .
)

```

### **EXITPRIOR, EXITPOST, and EXITREPLACE String...**

These keywords are optional. They specify a user exit program to run as part of the gather operation.

To specify an exit that is global to the Gather operation, specify EXITPRIOR or EXITPOST in the DATA clause. You can specify each of these keywords only once in the DATA clause. These keywords must come after all of the RULE clauses. EXITREPLACE cannot be used in the DATA clause.

You can also specify an exit that is specific to one RULE clause. Only one exit keyword is allowed in each RULE clause.

These keywords accept a list of strings separated by spaces. The first string is the name of the program to execute. The strings that follow are its parameters.

### **Using exit keywords in the DATA clause**

When used within a DATA clause, these keywords identify a program or command to be executed within a command shell. EXITPRIOR executes before all RULE statements have been processed; EXITPOST, after all RULE statements.

The exit keywords accept any executable file or command. The exit program must return an integer return value, with zero meaning the exit was successful.

## Using exit keywords in the RULE clause

EXITPRIOR, EXITPOST, and EXITREPLACE are optional within a RULE clause. Only one can be specified in any given RULE clause.

When used within a RULE clause, these keywords identify a program or command to be executed within a command shell before, after, or in place of processing of each Gather copy operation. The exit program is called once for each SOURCE specification entry within the SOURCE clause. Parameters are separated by spaces and passed to the exit in this order:

- Any parameters included in the invocation string
- The resolved SOURCE file specifications
- The resolved TARGET specification

The exit keyword accepts any executable file or command. The exit program must return an integer return value, zero meaning successful; it must also accept or ignore the additional Gather parameters added to the end of the invocation string.

When used in the context of the RULE clause, exit keywords must follow the TARGET keyword.

## Using exit keywords: an example

In the following example, the first EXITPRIOR statement relates to the DATA clause and specifies a user backup exit program, which executes before performing Gather copy operations. This backup exit is passed two flags. The command stream executed in an OS/2 shell is:

```
"e:\util\backup.cmd \i \t"
```

The second occurrence of the keyword illustrates how to use it in the context of a RULE clause. In this example, an encryption program will run against each source file specification. The exit program is passed the \k:347867 key option, the value for the source specification, and the value for the target specification. In this example, the command stream executed in an OS/2 shell is:

```
"encrypt \k:347867 d:\demoapp\a.exe f:\demoapp\bin":
```

The package file looks like this:

```
(DATA
  (PACKAGEFORMAT gather)
  (TARGETROOT d:\tcws)
  (RULE
    (SOURCE *.exe *.cmd)
    (TARGET exe)
    #|this program will be run for each source file|#
    (EXITPRIOR encrypt \k:347867 )
  )
  (EXITPRIOR "e:\util\backup.cmd \i \t" )
)
```

---

## Chapter 27. Using the NVBridge tool



The NVBridge tool is available only on OS/2.

The NVBridge tool supports automated distribution between a single NetView DM/2 CC server and its LAN-connected CC clients. It also supports remote distribution to APPC-connected NetView DM/2 servers and mainstream servers.

A sample build script named `nvbridge.cmd` is shipped with TeamConnection. It can be invoked within a TeamConnection builder. This build script maps TeamConnection build parameters to the command line syntax for invoking the NVBridge tool via the `teamcpak` command line interface.

You can use NVBridge as a builder for packaging in two ways:

- Integrate it with the gather step, so that the Gather tool leaves the package files in a directory from which NVBridge picks them up.
- Use it without the gather step. In this case, the build script for NVBridge must set up the directory and move files into it to interface correctly with the `teamcpak` command.

For information about setting up a build tree for running NVBridge, see “Setting up a build tree for the NVBridge tool” on page 328.

NVBridge produces the following NetView DM/2 output files:

### A change file

This file, containing all of the software deliverables, is stored in the `fsdata` subdirectory of the NetView DM/2 directory. The file name is *buildID.cf*, where *buildID* is the ID specified in the `-o "-b"` flag of the `teamcpak` command.

### A procedure file

This file is stored in the `fsdata` subdirectory of the NetView DM/2 directory. It contains the command instructions for uninstalling an installed software object during its next build. The file name is system-generated.

### A flat data file

This file, containing the text data of mail information that accompanies other objects, is stored in the `fsdata` subdirectory of the NetView DM/2 directory. The file name is system-generated.

### Catalog entries for the generated change file, procedure file, and mail notification object

These files are named according to the following naming convention:

`_Tx_corporation_ID.buildID.xxx.0.0`

Where:

- *x* is an identifier for the TeamConnection server. The default is C.
- *corporation\_ID* is a string of up to ten characters. The default is NULLCORP.

- *Build ID* is a string of up to 16 characters, representing the ID specified on the `-b` flag of the `teamcpak nvbridge` command.
- *xxx* identifies the file. The following values are used:
  - REF for the generated change file
  - CMD for the generated uninstall procedure
  - MAIL for the generated mail notification object

---

## Using the `teamcpak` command for NVBridge

To start the NVBridge tool, use the `teamcpak` command. This command is found in the directory where the TeamConnection family server is installed. If it is started from a build script, it does not need to be in the execution path of the machine from which the build is started.

The complete syntax for the `teamcpak nvbridge` command is the following. You must specify the command parameters in the order shown.

```
teamcpak -o "string" nvbridge input_source-directory
package_specification_file
```

### **-o "string"**

Specifies that the string listed in quotes be passed to the NVBridge tool. For a list of possible flags to be passed, see "Command line flags" on page 341 .

### **Input\_source\_directory**

A directory containing all the files and subdirectories of the software to be distributed using NetView DM/2. You must specify this directory as an absolute path with no wildcard characters.

The source root directory can be created in an earlier build step. The Gather tool can be used to create the source root directory and move the files into it as the TARGETROOT directory.

All files contained in the source root directory are included in the NetView DM/2 object that is built and distributed via NVBridge. It is important that the source root directory contain only the subdirectories and files required for the software package distribution.

File names within the source directory cannot contain blanks. Only non-hidden files are supported. The names of the files are reproduced on the target NetView DM/2 CC clients. Therefore, HPFS file names can be included only if the target NetView DM/2 CC clients listed in the package file have target drives that support HPFS.

If you are using the uninstall function, the uninstall program must reside in the source directory. If you use the MAIL keyword for remote servers, the MAIL text file that is sent to the remote servers must also reside within this directory.

### **package\_specification\_file**

A file describing how the NVBridge function is to build, catalog, and distribute software. Additional controls of NVBridge processing are provided through the optional command line flags described in the following section.

## Command line flags

You can specify the following flags in the `teamcpak` command, using the `-o` parameter. All of these flags except `-b` are optional.

### **-b:***buildID*

This flag is required. The build ID represents the software to be distributed. It is a string of up to 16 alphabetical characters or underscores. It cannot contain blanks or other special characters. If you are distributing software to NetView DM/2 clients whose target drives do not support HPFS file systems, the build ID can be only eight characters in length.

### **-v**

Use this flag to verify that NetView DM/2 CC clients listed in the `INSTALLS` keyword of the package file are defined to the NetView server and are in an active state. Verification takes place one time at the start of the `teamcpak` command. Changes that occur after verification are not detected.

Verification fails if one or more clients is in running rather than active status. Running status indicates that the client currently has a NetView command in progress.

If this flag is not specified, NVBridge ignores errors resulting from undefined or inactive clients.

If the `SENDS` keyword is specified in the package file, this flag also verifies the following:

- All of the remote destinations are defined in the NetView DM/2 remote destinations table.
- Any transmission queues associated with the remote destinations are in a released state.
- The transmission queues are empty.

If the `INSTALLS` or `SENDS` keywords are not specified in the package file, verification always succeeds.

### **-m**

Use this flag to monitor NVBridge-generated requests to NetView DM/2. If any install, uninstall, or send requests do not complete successfully, messages are generated.

This flag works with the `CLIENTINTERVAL`, `SENDINTERVAL`, and `ATTEMPTS` package file keywords. See "Writing a package file for NVBridge" on page 342 for details about using them to control monitoring durations and limits.

If you specify this flag, NVBridge continues to run as long as it takes to install on all the designated clients and send to remote destinations, or until the durations set by the package file keywords are exceeded.

If this flag is not specified, NVBridge submits the install and send requests and then ends without waiting for them to complete. In this case, you can use NetView DM/2 functions to track these pending requests.

### **-r**

Use this flag to retry an earlier failed attempt to install. Using the same package file, you can correct install failures without rebuilding the software object and without installing again on clients that were successful the previous time.

`TEST` and `SENDS` keyword in the package file are ignored, because testing and sending are assumed to have taken place already.

### **-a**

Use this flag to issue NetView `ACTIVATE` requests. This request causes the

client machine to reboot after installation processing and SEND requests are complete. This is the last request that NVBridge performs before completion.

This flag is ignored if the TEST and INSTALLS keywords are not specified in the package file.

If you specify this flag, you must also specify the `-m` flag.

- l** Use this flag to install software in a NetView DM/2 service area. This allows installation of files that replace system-locked files.

A NetView ACTIVATE request is required to move the files from the temporary service area and to make updates to the config.sys file take effect. You can use the `-a` flag to request the ACTIVATE. If you do not use the `-a` flag, you must use NetView to reboot the clients manually.

If you use this flag, the installation program must be installed on the client already, or the installation program must reside in the CC server's NetView DM/2 shared areas.

This flag is ignored if TEST and INSTALLS keywords are not specified in the package file.

- t:time** Use this flag to set a timer so that NVBridge runs at a later time. For example, you can invoke NVBridge, go home, and let it start during the night. This timer applies to all NVBridge functions, not individual INSTALLS or SENDS requests.

Valid values for time are 0000 to 2359. For example, specify `-t:0000` to run NVBridge starting at midnight. Specify `-t:1200` to start NVBridge at noon.

- f** Use this flag to prevent the sending of objects that have install or uninstall problems. If any INSTALL or UNINSTALL requests fail, NVBridge returns a nonzero return code and purges any pending requests.

If this flag is not specified, NVBridge returns a return code of 0 and continues with SENDS requests, even if some UNINSTALL or INSTALL requests fail to complete.

## Examples of the teamcpak nvbridge command

The following is an example of the teamcpak nvbridge command.

```
teamcpak -o "-b:demoapp -m -v" nvbridge d:\demoapp demoapp.pkf
```

The input source directory, d:\demoapp, contains the files to be installed. The file demoapp.pkf is the package specification file.

The `-o` parameter passes three flags to the NVBridge tool:

- `-b:demoapp` specifies that demoapp is the build ID.
- `-m` specifies that NVBridge monitor requests to NetView DM/2 to completion.
- `-v` specifies that NVBridge verifies that all clients are defined to NetView DM/2 and active.

---

## Writing a package file for NVBridge

This section describes the NetView DM/2 package file keywords and their effect on normal processing behavior.

A sample package file named `nvbridge.pkf` is shipped with TeamConnection. You can customize it for your own use.

## Syntax rules for an NVBridge package file

Follow these syntax rules when you write a package file:

- Package files are free format. Text is not positional, and many statements can exist on the same line.
- Comments can appear anywhere within the file. Use the characters `#|` and `|#` as delimiters, as shown in the following example:  
`#| This is a comment |#`
- Package file keywords must be prefixed with a left parenthesis and must have a corresponding balanced right parenthesis to end the scope of the keyword.
- If the value for a keyword is a string that contains blanks or parentheses, enclose the string in double quotes.

The following shows the syntax of a package file for NVBridge. The order of the keywords inside the NVGLOBALS clause does not matter; all other keywords must appear in the order shown. You must supply the values for the strings that are shown in *italics*.

```
(DATA
  (PACKAGEFORMAT nvbridge
  (NVGLOBALS
    [(MAIL filename)]
    [(TEAMCSERV x)]
    [(CORPID name)]
    (INSTALLDIR path)
    [(INSTALLPGM path\filename)]
    [(IPARMS parameters)]
    [(UNINSTALLPGM filename)]
    [(CLIENTINTERVAL n)]
    [(SENDINTERVAL n)]
    [(ATTEMPTS n)]
  )
  [(TEST
    (ENTRY client)
  )]
  [(INSTALLS
    (ENTRY client target_directory)
    .
    .
  )]
  [(SENDS
    destination
    .
    .
  )]
)
```

## Keywords for an NVBridge package file

**DATA** This keyword is required. It must be the first keyword in the package file, and it can be specified only once.

All other keywords are nested within the DATA clause.

### Example:

```
(DATA
  .
  .
```

```

        other keywords go here
        .
    )

```

### **PACKAGEFORMAT nvbridge**

This required keyword must be the first keyword within the DATA clause. It can be specified only once. It tells the teamcpak command that this package file is for NVBridge.

#### **Example:**

```

(DATA
    .
    .
    (PACKAGEFORMAT nvbridge)
    .
)

```

### **NVGLOBALS**

This required keyword must follow the PACKAGEFORMAT keyword within the DATA clause. It can be specified only one time.

All the global and default keywords for the package file are specified in the NVGLOBALS clause. At least one INSTALLDIR keyword must be specified within this clause; all other NVGLOBALS keywords are optional.

Keywords in the NVGLOBALS clause can appear in any order.

#### **Example:**

```

(DATA
    .
    .
    (NVGLOBALS
        .
        NVGLOBALS keywords go here
        .
    )
    .
)

```

### **MAIL *filename***

This optional keyword can appear at any place within the NVGLOBALS clause. It can be specified only one time.

This keyword identifies a text file that is sent as a mail notification object to all destinations listed in the SENDS keyword. The file name specified must contain only the file name and extension, no path information. NVBridge searches the input directory specified on the command and uses the first file that matches this file name.

The mail object is created and cataloged on the local server along with the .ref and .cmd objects. Mail objects typically take the form of readme files.

#### **Example:**

In the following example, the input directory is searched for the first occurrence of the file readme.txt during NetView DM/2 processing. This file is used to create a new .mail object cataloged to NetView DM/2. This object is then sent along with the .ref and .cmd objects during the processing of any SENDS requests.

```

(DATA
    .
    .
    (NVGLOBALS

```



```

        .
        (MAIL  readme.txt )
    .
)
.
.
)

```

#### **TEAMCSERV *x***

This optional keyword can appear at any place within the NVGLOBALS clause. It can be specified only one time.

This keyword specifies a 1-character alphanumeric ID to be used in global name generation. It identifies the TeamConnection family server that built and distributed the global named object. If this keyword is specified, the first four characters of the global name are `_Tx_`. If this keyword is not specified, the first characters of the global name default to `_TC_`.

#### **Example:**

An example of a NetView DM/2 object name based on the following example is `_t1_nullcorp.demoapp.ref.0.0`.

```

(DATA
.
.
(NVGLOBALS
.
(TEAMCSERV 1 )
.
)
.
.
)

```

#### **CORPID *name***

This optional keyword can appear at any place within the NVGLOBALS clause. It can be specified only one time.

This keyword is used in constructing the global names of NetView objects generated by NVBridge. You can use it to further identify the NetView objects, for example by specifying the name of your company. The name specified in this keyword follows the value of the TEAMCSERV keyword in global names.

The name specified in this keyword can be from one to ten alphanumeric characters. If this keyword is not specified, it defaults to the value NULLCORP.

#### **Example:**

An example of a NetView DM/2 object name based on the following example is `_t1_ibmcorp.demoapp.ref.0.0`.

```

(DATA
.
.
(NVGLOBALS
.
(CORPID  ibmcorp)
.
)
.
.
)

```

**INSTALLDIR *path***

This required keyword can appear at any place within the NVGLOBALS clause. It can be specified only one time.

This keyword defines the default workstation directory that is to be created as the root software directory on the clients.

The *path* must be an absolute file specification, including a valid drive. You can override this value for individual clients, within the ENTRY clause of the INSTALLS keyword. This target directory need not exist on the CC clients. NetView creates this directory and populates it with the same data as contained in the input directory specified on the teamcpak command.

**Example:**

In the following example, the path e:\demoapp is used as the target drive and directory for all CC clients listed in the INSTALLS keyword. NetView DM/2 will install the files and subdirectory structure specified on the teamcpak command within a target directory e:\demoapp on each CC client machine.

```
(DATA
  .
  .
  (NVGLOBALS
    .
    (INSTALLDIR e:\demoapp )
    .
  )
  .
  .
)
```

**INSTALLPGM *path\filename***

This optional keyword can appear at any place within the NVGLOBALS clause. It can be specified only one time.

This keyword defines the installation program to be invoked after the software files are copied on the CC client machine. If this keyword is not specified, the installation process includes only the movement of files, and an installation program is not invoked.

The installation program can be specific to the software just installed. In this case, the installation program is itself a file that was copied. Alternatively, the installation program can be a separate software product installed previously on the client machine. The installation program must meet the following criteria:

- It must be in the client's execution path, or the executable name must include its fully qualified path.
- The return value of the program must be 0 if you want NetView DM/2 to catalog the installation as successful. If the program returns a value other than 0, NetView DM/2 concludes the installation was a failure.
- The installation program cannot be interactive; that is, it cannot expect user input.

The installation program can invoke other programs.

The INSTALLS entries and hence the path to the installation program can differ from client to client. As a result, the installation program often needs to know the target directory into which the software was installed. To aid with NetView environmental information such as this, you can include the



```

        (IPARMS  "\i \t \g \d:${TargetDir}" )
    .
)
.
)

```

#### **UNINSTALLPGM** *filename*

This optional keyword can appear at any place within the NVGLOBALS clause. It can be specified only one time.

This keyword defines the program that will be used to uninstall this version of the software when its next version is installed. For example, if you are installing version 1.0 of your package, use this keyword to specify the program that will be used to uninstall version 1.0 when version 1.1 is installed.

The uninstall program must be contained in the input software directory. NVBridge searches the input directory specified on the teamcpak command and uses the first file that matches the file name specified on this keyword.

If both this keyword and the INSTALLS or TEST keywords are specified, then a NetView PROCEDURE object is created along with the software object. When the next version of this package is installed, this uninstall procedure is run against previous install clients, before the removal of the old version of the software and the creation of the new NetView software object. NVBridge tracks only whether the uninstall operations complete, not whether they are successful.

The uninstall program can itself invoke other programs.

#### **Example:**

In the following example, the input directory is searched for the first occurrence of the file `uninst.cmd` during NetView DM/2 processing. This file is used to create a new uninstall PROCEDURE object cataloged to NetView DM/2.

```

(DATA
.
.
(NVGLOBALS
.
  (UNINSTALLPGM  uninst.cmd )
.
)
.
)

```

#### **CLIENTINTERVAL** *n*

This optional keyword can appear at any place within the NVGLOBALS clause. It can be specified only one time.

This keyword identifies the sleep interval used for monitoring client operations such as uninstalling and installing. It is used with the `-m` flag on the teamcpak command.

Specify a value in seconds. Valid values are from 10 to 600. The default is 15 seconds.

#### **Example:**

```

(DATA
.
.

```

```

(NVGLOBS
.
(CLIENTINTERVAL 60 )
.
)
.
.
)

```

### **SENDINTERVAL *n***

This optional keyword can appear at any place within the NVGLOBS clause. It can be specified only one time.

This keyword identifies the sleep interval used for monitoring SEND operations. It is used with the `-m` flag on the `teamcpak` command.

Specify a value in seconds. Valid values are from 10 to 600. The default is 15 seconds.

#### **Example:**

```

(DATA
.
.
(NVGLOBS
.
(SENDINTERVAL 40 )
.
)
.
.
)

```

### **ATTEMPTS *n***

This optional keyword can appear at any place within the NVGLOBS clause. It can be specified only one time.

This keyword identifies the maximum number of attempts to monitor NVBridge operations. Specify a number from 1 to 6. The default is 4.

This value is combined with the CLIENTINTERVAL and SENDINTERVAL values to compute the maximum monitoring time. For each operation per client or prior remote destination, NVBridge uses this formula:

Monitor and check every *x* seconds up to *y* times

Where *x* is the value for CLIENTINTERVAL or SENDINTERVAL, and *y* is the value for ATTEMPTS.

#### **Example:**

If you are installing to four clients and you specify the `-m` flag on the `teamcpak` command, by default NVBridge monitors every 15 seconds, up to four attempts. It repeats this 4 times for each of the four clients, resulting in a total of 16 attempts at 15-second intervals.

```

(DATA
.
.
(NVGLOBS
.
(ATTEMPTS 4 )
.
)

```

```
)  
:  
.)
```

**TEST** This optional keyword appears within the DATA clause before the INSTALLS or SENDS keywords. It can be specified only one time.

This keyword identifies a single NetView DM/2 CC client to be used as a test machine. All normal processing is first performed against this single client. If everything succeeds, normal processing continues for all clients listed in the INSTALLS keyword; otherwise normal processing stops.

This keyword accepts a single ENTRY keyword, which identifies the client. Do not repeat this client value in the INSTALLS entries, or NVBridge might fail.

**Example:**

In the following example, the test CC client named CLIENT1 will be used to perform a test installation. The software and data will be installed to the target client directory e:\demoapp.

```
(DATA  
.  
.  
  (TEST  
    (ENTRY CLIENT1 e:\demoapp)  
  )  
.  
.)
```

**INSTALLS**

This optional keyword is specified within the scope of the DATA clause. It must follow the NVGLOBALS and TEST keywords. It can be specified only one time.

This keyword identifies the list of ENTRY keywords specifying the clients where the software object is to be installed. Duplicate ENTRY keywords for the same client are ignored.

**ENTRY** *client, directory*

This required keyword is specified within the scope of the TEST or INSTALLS clauses. It can be specified many times.

This keyword identifies a NetView DM/2-defined CC client workstation on which the software is to be installed. For each ENTRY keyword, you must specify the name of a CC client machine.

Optionally, you can also specify a target installation directory for the client. This directory overrides the directory specified in the INSTALLDIR keyword. The target directory must be an absolute file specification, including a valid drive. If this value is found to be not valid, NetView uses the default value found in the INSTALLDIR keyword.

**Example:**

In the following example, the .ref object created by NetView DM/2 will be installed to CLIENT1, CLIENT2, CLIENT3, and CLIENT4 CC client machines. In the case of CLIENT2 and CLIENT4, the software is installed in the default INSTALLDIR value of d:\demoapp. For the others, the target directory in the ENTRY keyword overrides the INSTALLDIR value.

```

(DATA
  .
  (NVGLOBALS
    .
    (INSTALLDIR d:\demoapp )
    .
  )
  .
  (INSTALLS
    (ENTRY CLIENT1 e:\demoapp)
    (ENTRY CLIENT2 )
    (ENTRY CLIENT3 c:\demoapp)
    (ENTRY CLIENT4 )
  )
  .
  .
)

```

#### **SENDS** *destination ...*

This optional keyword is specified within the scope of the DATA clause. It must follow the NVGLOBALS, TEST, and INSTALLS keywords. It can be specified only one time.

This keyword identifies the list of remote destinations that are to receive NVBridge-created objects. These destinations are APPC-connected NetView DM family servers.

Each remote destination in this list should be configured to accept creation or replacement of cataloged objects. If the remote server does not allow incoming SENDS of objects, then NVBridge cannot send objects to it. Also, if the remote server accepts only creates and not replacements, then NVBridge can send it only objects that do not already exist in its catalog.

#### **Example:**

```

(DATA
  .
  .
  (SENDS
    USSNANR.AUSTIN2
    USSNANR.AUSTIN3
    USSNANR.NEWYORK1
    USSNANR.CHICAGO4
  )
  .
  .
)

```

---

## **Problem determination for NVBridge**

If a particular object has a status of SCHEDULED or IN PROGRESS that does not reflect its true status, then the existing version of the software object might be in a bad locked state. The result is that NetView DM/2 cannot build new versions of the object. NetView DM/2 always attempts to purge all previous locked requests when it builds new versions of software to be distributed. However, there are abnormal NetView DM/2 cases where locked objects require further manual intervention to correct locked NetView DM/2 files.

If during NetView DM/2 processing, messages indicate that NetView DM/2 failed because it could not remove a previous version of an object, try the following steps. If these steps fail to correct the problem, then contact your IBM NetView DM/2 representative.

1. Check the NetView DM/2 message.dat file on the server and, if possible, on the client, checking to see if any NetView DM/2 errors have been detected. If a NetView DM/2 error has occurred, then take appropriate steps to report and correct the problem.

Rebooting your system sometimes will temporarily correct the NetView DM/2 condition so that you can get your work done. Even if you take this route, go on to the next steps after rebooting.

2. Look at the request queue contents and at the install history to find the object name generated from NetView DM/2 processing. The name of the NetView DM/2 object is also in the messages. The locked entries can be identified by inspecting the NetView DM/2 request queue and seeing an entry for the object, where it is obvious that the entry is not being processed. At other times, a locked entry can be found by looking at the install history for clients that have install status other than INRU.
3. Temporarily undefine from the server the CC clients that involve the lock. Locked requests for undefined clients can sometimes be corrected by making the CC client unknown to the CC server.  
This step causes the NetView DM/2 CC client to stop running. Restart the client either manually or via another remote access to the machine.
4. If any requests are in the request queue, try to both purge and delete the request. (If you are using the GUI, the following steps can be combined into one or two steps.)
5. Perform a Deleteit against the object's group name to remove install target information that might have been set previously. Do this for all client workstations with the /ws option.
6. Remove all install history for the object.
7. Redefine to the server the CC clients that were locked.
8. Try your initial NetView DM/2 request again.

---

## NVBridge utilities

TeamConnection provides a collection of utilities that can be combined into a user-defined build script, to help automate customized forms of packaging steps. You can use these utilities instead of the teamcpak command, to customize your distribution steps.

**Note:** These interfaces are program-sensitive interfaces used by NVBridge. As a result, these interfaces are likely to change and evolve from release to release, and no IBM commitment is implied that these interfaces will remain unchanged and compatible in future versions.

The only form of parameter checking performed by these utilities is verifying that required parameters are specified and that the parameter list meets syntax requirements. These utilities assume that valid parameter values are passed to them.

To display the syntax of these utilities, type the name followed by a question mark. For example, type FHP0BDEL ? to see the syntax of this tool.



## FHPSTAT

Use this function to check the status of the NetView DM/2 CC server components. This is used to determine if NetView DM/2 is available and if the necessary components have been started.

This utility continues until it has made the number of attempts specified in the *loop* parameter or until the return code is 0, whichever occurs first.

### Syntax

```
fhpstat [timer], [loop], [config=lan|APPC], [display=YES|no]
```

- *timer* - time in seconds for sleep interval for monitoring.
- *loop* - number of times to loop before giving up. Defaults to 1 attempt, meaning it will not loop and monitor.
- *config* - LAN means to check only for Agent and Change Controller components, whereas APPC means to check also for Transmission Controller.
- *display* - specifies whether to display results or not.

### Return Codes

- Returns 0 if the CC server components are running. What components have to be running depends on standalone LAN or APPC environment.
- Returns 1 if one or more CC server components are not running.
- Return 4 if not an APPC configured CC server.
- Return 8 if not a NetView DM/2 CC server machine.

### Example

```
fhpstat 30, 2, lan, yes
```

## FHPOBDEL

This function unconditionally attempts to remove all NetView DM/2 information about a cataloged NetView DM/2 object, including anything currently in the request queue. Deletes all component name information and any previous NetView DM/2 ADDIT requests that might be in effect for the object.

### Syntax

```
fhpobdel object, [display=YES|no]
```

- *object* - NvDM/2 global object name.
- *display* - specifies whether to display results or not.

### Return Codes

- Returns 0 if the syntax of FHPOBEL was valid. Check for success using the FHPISCAT tool.
- Return 8 if a parameter error occurred or object name was not specified.

### Example

```
fhpobdel _tc_demoapp.nullcorp.ref.0.0, no
```

## FHPOBMON

Use this function to check the install history for a particular NetView DM/2 global object against an input list of install clients listed in an input file *inclients*.

This tool continues until it has made the number of attempts specified in the *loop* parameter or until the return code is 0, whichever occurs first.

This function only supports installation monitoring related to CC clients, not to the local CC server.

## Syntax

```
fhpbmon object, inclients, [timer], [loop], [display=YES|no]
```

- *object* - NetView DM/2 global object name.
- *inclients* - the name of the file containing the list of CC clients.
- *timer* - time in seconds for sleep interval for monitoring.
- *loop* - number of times to loop before giving up. Defaults to 1 attempt, meaning it will not loop and monitor.
- *display* - specifies whether to display results or not.

## Return Codes

- Returns 0 if an *object* existed and all clients listed in *inclients* file were installed. The *inclients* file is emptied.
- Returns 1 if an *object* existed but some of the clients in the *inclients* were not reflected in the object's install history. Input file is updated with the list of only those clients in *inclients* originally that did not show up in the history of the object.
- Returns 2 if an *object* did not exist in CC server catalog. The input file is unchanged.
- Returns 8 if a parameter error occurred on invocation. The input file is unchanged.
- Returns 16 if an internal error occurred. The input file is unchanged.

## Example

```
fhpbmon _tc_demoapp.nullcorp.ref.0.0, infile, 30, 2, yes
```

Each line of the file *infile* must contain a client name. No blank lines are allowed.

```
client1
client2
client3
beta1
test1
:
```

## FHPOBDIF

Use this function to cross-reference the differences between the install history of a NetView DM/2 global object and a particular list of install clients listed in an input file. All clients in *file1* that are not in the install history are left in *file1*. All clients in the install history that are not in *file1* are put into *file2*. Hence *file1* is updated with the list of clients that should be installed, that are not already installed, and *file2* is updated with the list of clients that are installed, but are not in the list of clients that should be installed. This command generates the differences.

This function only supports install history related to CC clients, not the local CC server.

## Syntax

`fhpbodif object, file1, file2, [display=YES|no]`

- *object* - NetView DM/2 global object name.
- *file1* - input file of comparison clients, also output file for clients that were not in the install history.
- *file2* - output file updated with clients that were in the install history but which were not in the original *file1* list of clients.
- *display* - specifies whether to display results or not.

## Return Codes

- Returns 0 if an object existed and there were no differences. *file1* and *file2* are emptied.
- Returns 1 if an object existed but there were differences. *file1* and *file2* reflect these differences.
- Returns 2 if an object did not exist. *file1* and *file2* are unchanged.
- Returns 8 if a parameter error occurred on invocation. *file1* and *file2* are unchanged.
- Returns 16 if an internal error occurred. *file1* and *file2* are unchanged.

## Example

```
fhpbodif _tc_demoapp.nullcorp.ref.0.0, file1, file2, no
```

Each line of the file *file1* must contain a client name. No blank lines are allowed.

```
client1
client2
client3
beta1
test1
:
:
```

## FHPISCAT

This function checks to see if a NetView DM/2 object exists in the NetView DM/2 catalog. It checks only for a catalog entry, not the associated file.

## Syntax

`fhpiscat object, [display=YES|no]`

- *object* - NetView DM/2 global object name.
- *display* - specifies whether to display results or not.

## Return Codes

- Returns 0 if an object was cataloged.
- Returns 1 if the object was not cataloged.
- Return 8 if a parameter error occurred and object name was not specified.

## Example

```
fhpiscat _tc_demoapp.nullcorp.ref.0.0
```

## FHPICAT

This function does the following:

- Creates a NetView DM/2 software object based on the name specified by *object*

- Walks the input directory *sdir* to create the filespeclist section of the object
- Names the change file based on *buildid*
- Sets the InstallDir value in the profile to the value of *tdir*
- Sets the InstallSection of the profile to reference the install program of *ipgm* and parms of *iparms*

### Syntax

```
fhpicat object, tdir, sdir, buildid, [ipgm], [iparm], [display=YES|no]
```

- *object* - NetView DM/2 global object name.
- *tdir* - value to be used for TargetDir in object change profile.
- *sdir* - fully qualified path of input directory where software resides.
- *buildID* - value to be used in naming the change file created and stored in NetView DM/2 fsdata subdirectory.
- *ipgm* - name of an installation program to be used for installation.
- *iparm* - value of installation program parameters to be specified in the object profile.
- *display* - specifies whether to display results or not.

### Return Codes

- Returns 0 if a new object create attempt was successful. FHPISCAT can be used to verify that the object was created.
- Returns 1 if a new object create attempt failed.
- Returns 8 if a parameter error occurred.

### Example

```
fhpicat _tc_demoapp.nullcorp.ref.0.0, d:\demoapp, c:\demoapp,  
$(targetdir)\inst.exe, -i -f -t, yes
```

## FHPUCAT

This function does the following:

- Creates a NetView DM/2 PROC (procedure) object based on the name specified by *object*.
- Walks the input directory *sdir*, looking for the first occurrence of the *procedure* file.
- Copies the file into the NetView DM/2 fsdata subdirectory.
- Catalogs the object to NetView DM/2. This object can then have NetView DM/2 INITIATE commands requested against it.

### Syntax

```
fhpucat object, procedure, sdir, [display=YES|no]
```

- *object* - NetView DM/2 global object name.
- *procedure* - the file name of the REXX procedure file located within the *sdir* that is to be used to create the object.
- *sdir* - fully qualified path of input directory where software resides.
- *display* - specifies whether to display results or not.

### Return Codes

- Returns 0 if a new object create attempt was successful. FHPISCAT can be used to verify that the object was created.
- Returns 1 if a new object create attempt failed.

- Returns 8 if a parameter error occurred.

### Example

```
fhpucat _tc_demoapp.nullcorp.proc.0.0, uninst.cmd, c:\demoapp, no
```

## FHPMCAT

This function does the following:

- Creates a NetView DM/2 flat data text object based on the name specified by *object*.
- Walks the input directory *sdir*, looking for the first occurrence of the *mail\_file* file name.
- Copies the file into the NetView DM/2 fsdata subdirectory.
- Catalogs the object to NetView DM/2. This object can then have NetView DM/2 SEND commands requested against it, or its contents can be viewed.

### Syntax

```
fhpmdat object, mail_file, sdir, [display=YES|no]
```

- *object* - NetView DM/2 global object name.
- *mail\_file* - the file name of a text file located within the *sdir* specified that is to be used to create the object.
- *sdir* - fully qualified path of input directory where software resides.
- *display* - specifies whether to display results or not.

### Return Codes

- Returns 0 if a new object create attempt was successful. FHPISCAT can be used to verify that the object was created.
- Returns 1 if a new object create attempt failed.
- Returns 8 if a parameter error occurred.

### Example

```
fhpmdat _tc_demoapp.nullcorp.txt.0.0, readme.txt, c:\demoapp, no
```

## FHPVERIF

This function verifies that the CC clients listed in the *inclients* file are defined to the local CC server and in active status. The file *inclients* is then modified to contain only those clients that were not defined or active, or it is emptied if all clients are verified.

This tool continues until it has made the number of attempts specified in the *loop* parameter or until the return code is 0, whichever occurs first.

### Syntax

```
fhpverif inclients, timer, loop, [display=YES|no]
```

- *inclients* - the name of the input file containing the list of CC client names to be verified. This file will be modified to contain the list of those clients that failed to verify.
- *timer* - time in seconds for sleep interval for monitoring.
- *loop* - number of times to loop before giving up. Defaults to 1 attempt, meaning it will not loop and monitor.
- *display* - specifies whether to display results or not.

## Return Codes

- Returns 0 if every client in the file was defined and active to local CC server.
- Returns 1 if one or more clients in the file did not verify.
- Returns 8 if a parameter error occurred or an input file *inclients* did not exist.

## Example

```
fhpverif infile1.inp, 30, 2
```

Each line in the file (infile1.inp) must contain a client name. No blank lines are allowed.

```
client1
client2
client3
beta1
test1
:
:
```

## FHPRQPUR

This function purges any outstanding requests in the NetView DM/2 request queue related to the *object*. After the function completes, you can use the FHPRQMON tool to confirm that no *object*-related requests are in the request queue.

This function supports only object-related requests made to CC clients, not the local CC server. In other words, if there is a request for this *object* but it is for the CC server, then the request will not be detected or purged.

## Syntax

```
fhprqpur object, forced=yes|NO, [display=YES|no]
```

- *object* - NetView DM/2 global object name to be purged from request queue.
- forced - a yes or no value that defaults to no. If yes is specified, this tool will use all means possible to force the purging of related requests. If no is specified, this tool will try only normal means to purge related requests. A forced purge can result in requests being purged in the middle of processing.
- display - specifies whether to display results or not.

## Return Codes

- Returns 0 if one or more requests for the specified *object* was found in the request queue, and a purge attempted.
- Returns 1 if no matching requests were found in the request queue.
- Returns 8 if a parameter error occurred.
- Returns 16 if an internal processing error occurred.

## Example

```
fhprqpur _tc_demoapp.nullcorp.ref.0.0, yes, yes
```

## FHPRQMON

This function monitors any outstanding requests in the NetView DM/2 request queue related to the *object*.

This tool continues until it has made the number of attempts specified in the *loop* parameter or until the return code is 0, whichever occurs first.

This function only supports object-related requests made to CC clients, not the local CC server.

### Syntax

`fhprqmon object, file, [timer], [loop], [display=YES|no]`

- *object* - NetView DM/2 global object name to be monitored from request queue.
- *file* - an output file to be created/updated to contain the list of request identifiers that are in the request queue regarding the particular object.
- *timer* - time in seconds for sleep interval for monitoring.
- *loop* - number of times to loop before giving up. Defaults to 1 attempt, meaning it will not loop and monitor.
- *display* - specifies whether to display results or not.

### Return Codes

- Returns 0 if there are no outstanding requests for the *object*.
- Returns 1 if there were requests detected in the queue and *file* has the list of their request IDs.
- Returns 8 if a parameter error occurred.
- Returns 16 if an internal processing error occurred.

### Example

```
fhprqmon _tc_demoapp.nullcorp.ref.0.0, outfile, 0, 1
```

The output in the file outfile would look like the following:

```
3
4
16
⋮
```

## FHPTRVER

This function verifies that every NetView DM/2 remote destination listed in the input file *indests* is defined to the local CC server and that all the related NetView DM/2 transmission queues are empty and in a released state.

This tool continues until it has made the number of attempts specified in the *loop* parameter or until the return code is 0, whichever occurs first.

### Syntax

`fhptrver indests, timer, loop, [display=YES|no]`

- *indests* - the name of the input file containing the list of NetView DM/2 remote destination names to be verified. This file will be modified to contain the list of those destinations that failed to verify, or the file will be empty if they all verified.
- *timer* - time in seconds for sleep interval for monitoring.
- *loop* - number of times to loop before giving up. Defaults to 1 attempt, meaning it will not loop and monitor.
- *display* - specifies whether to display results or not.

### Return Codes

- Returns 0 if every remote destination in the file was defined and related transmission queue was empty and in a released state.
- Returns 1 if one or more remote destinations did not verify.
- Returns 8 if a parameter error occurred or an input file *indests* did not exist.

- Returns 16 if an internal processing error occurred.

### Example

```
fhptrver infile1.inp, 30, 2
```

Each line of `infile1.inp` must contain a remote destination name. No blank lines are allowed.

```
usibmnr.austin1
usibmnr.austin2
usibmnr.newyork3
usibmnr.newyork6
usibmnr.chicago1
:
:
```

## FHPTRPUR

This function empties the transmission queues and releases the transmission queues corresponding to the list of remote destination names specified in the input file *indests*.

### Syntax

```
fhptrpur indests, [display=YES|no]
```

- *indests* - the name of the input file containing the list of NetView DM/2 remote destination names to be purged. This file will be modified based on the return codes.
- *display* - specifies whether to display results or not.

### Return Codes

- Returns 0 if no purges were performed due to all transmission queues already being empty and released. Input file is emptied.
- Returns 1 if some purge modifications were performed against transmission queues. The input file is updated to reflect that remote destinations triggered purge requests.
- Returns 8 if a parameter error occurred or there is a missing input file.
- Returns 16 if an internal processing error occurred.

### Example

```
fhptrpur infile1
```

Each line of `infile1` must contain a remote destination name. No blank lines are allowed.

```
usibmnr.austin1
usibmnr.austin2
usibmnr.newyork3
usibmnr.newyork6
usibmnr.chicago1
:
:
```



---

## Chapter 28. Using the Tivoli Software Distribution packaging tool

The Tivoli Software Distribution packaging tool supports automated distribution between a single Tivoli Software Distribution server and its TCP/IP-connected clients. The Tivoli Software Distribution tool works either by itself or in conjunction with TeamConnection's Gather tool to enable you to distribute files through Tivoli Software Distribution. Use of this tool requires you to be familiar with Tivoli configuration and system administration so that TeamConnection can start Tivoli Software Distribution to distribute file packages.

The Tivoli Software Distribution distribution tool must be run on a Tivoli managed node running on any of TeamConnection's UNIX platforms or Windows NT.

The Tivoli Software Distribution distribution tool includes a sample build script named `softdist` (on UNIX platforms) or `softdist.exe` (on Windows/NT). It can be run from within a TeamConnection builder. This build script maps TeamConnection build parameters to the command line syntax for the Tivoli Software Distribution tool through the `teamcpak` command line interface.

You can use Tivoli Software Distribution as a builder for packaging in two ways:

- Integrate it with the gather step, so that the Gather tool leaves the package files in a directory from which Tivoli Software Distribution picks them up.
- Use it without the gather step. In this case, the build script for Tivoli Software Distribution must set up the directory and move files into it to interface correctly with the `teamcpak` command.

To simplify the interface, the Tivoli Software Distribution tool uses a select set of options. If you want to take full advantage of Tivoli Software Distribution features, you can import a Tivoli Software Distribution package specification. Importing a package specification provides you access to all Tivoli Software Distribution functions.

The Tivoli Software Distribution tool produces a Tivoli FilePackage, which is used for distribution.

---

### Using the `teamcpak` command with Tivoli Software Distribution

To start the Tivoli Software Distribution tool, use the `teamcpak` command. This command is found in the directory where the TeamConnection family server is installed. If it is started from a build script, it needs to be in the execution path of the build server.

The complete syntax of the `teamcpak softdist` command follows. You must specify the command parameters in the order shown.

```
teamcpak [-i] [-o "string"] softdist inputFile
```

**-i** Specifies that only one *inputFile* is specified in the command: an include file containing the list of input files. This parameter is optional.

**-o "string"**

Specifies that the string listed in quotes be passed to the Tivoli Software Distribution tool. For a list of possible flags to be passed, see "Command line flags" on page 362.

### inputFile

Specifies the files to be copied and the name of the package specification file. You can specify this parameter in these ways:

- Specify the name of an include file, whose contents is a list of input files. One of these input files must be a package specification file with the extension .pkf. In this case, you must also specify the -i parameter.
- Specify a list of two or more files. One of the files must be a package specification file with the extension .pkf.
- Specify the directory from which the files are to be copied and the name of the package specification file.

If more than one package file is listed, the first package file on the command line or in the include file is used, and the others are treated as ordinary files.

The following are examples of specifying input files.

```
teamcpak -i softdist myInputFile
teamcpak softdist inputFile1 inputFile2 inputFile3 . . .
teamcpak softdist d:\inputDir\myPkfFile.pkf
```

## Command line flags

You can specify the following flags in the teamcpak command, using the -o parameter. All of these flags are optional.

- a** Create directories on the target.
- c** Clear the target (delete all specified files and directories) before the apply. If you use this option, do not use the -x option.
- t** Overwrite existing files (delete files on the target prior to distribution).
- m** Accept input errors, such as missing files and directories from the SOURCE keyword.
- n** Send no notices to Tivoli. If you want to post Tivoli notices, you must configure Tivoli Notices before using this packaging tool.
- p** Preview only; do not actually distribute files.
- r** Reboot the target after distribution.
- x** If an error occurs, leave any distributed files on the target; do not clean up. If you use this option, do not use the -c option.
- k** Keep the Tivoli file package. To enable the Tivoli Software Distribution tool to perform more efficiently, the Tivoli Software Distribution package file is created when the package part is created and then destroyed and recreated whenever the part is modified. Use the -k option to prevent the package file from being destroyed.

## Example of the teamcpak softdist command

The following is an example of the teamcpak softdist command.

```
teamcpak -i -o "-a -n -t" softdist Client.lst
```

The -i parameter specifies that the input file Client.lst is to be used. The -o parameter passes the following options to Tivoli Software Distribution:

- -a creates directories on the target.

- -n indicates that no error notices are to be sent to Tivoli Software Distribution.
- -t indicates that any existing files on the target are to be overwritten.

---

## Writing a package file for Tivoli Software Distribution

This section describes the Tivoli Software Distribution package file keywords and their effect on normal processing behavior.

A sample package file named `client.pkf` is shipped with TeamConnection. You can customize it for your own use.

### Syntax rules for a Tivoli Software Distribution package file

Follow these syntax rules when you write a package file:

- Package file keywords must appear in the order shown below.
- Package file keywords must be prefixed with a left parenthesis and must have a corresponding balanced right parenthesis to end the scope of the keyword.
- If the value for a keyword is a string that contains blanks or parentheses, enclose the string in double quotes.
- Default options are supplied for all Tivoli Software Distribution required Tivoli Software Distribution options. Specific options can be set in your TeamConnection package file.
- Comments can appear anywhere within the file. Use the characters `#|` and `|#` as delimiters, as shown in the following example:

```
#| This is a comment |#
```

The following shows the syntax of a package file for Tivoli Software Distribution. The keywords must appear in the order shown here. You must supply the values for the strings that are shown in *italics*.

```
(DATA
(PACKAGEFORMAT softdist)
(TARGETROOT filename)
(MANAGER ProfileManager)
(NODES "ManagedNode... PCManagedNode...")
(IMPORT filename |
  [(DISTRIBUTE [FULL | CHANGED])
  [(INSTALLPGM filename)]
  [(LOGNODE ManagedNode)]
  [(LOGFILE directory)]
)
)
```

### Keywords for a Tivoli Software Distribution package file

**DATA** This keyword is required. It must be the first keyword in the package file, and it can be specified only once.

All other keywords are nested within the DATA clause.

**Example:**

```
(DATA
.
.
other keywords go here
.
.
)
```

### **PACKAGEFORMAT softdist**

This required keyword must be the first keyword within the DATA clause. It can be specified only once. It tells the teamcpak command that this package file is for Tivoli Software Distribution.

#### **Example:**

```
(DATA
.
.
(PACKAGEFORMAT softdist)
.
.
```

### **TARGETROOT**

This keyword specifies the directory path to which files are to be distributed on the target systems. You can specify only one target root. All target systems use identical target roots.

#### **Example:**

```
(DATA
.
.
(TARGETROOT /usr/local/teamc/images)
.
.
```

### **MANAGER**

This keyword specifies a Tivoli Software Distribution profile manager that you have already created in the Tivoli Software Distribution system.

#### **Example:**

```
(DATA
.
.
(MANAGER Distrib1)
.
.
```

### **NODES**

This keyword specifies the nodes to which the files are to be distributed. These must already have been defined to the profile manager as subscriber ManagedNodes or PCManagedNodes. To distribute files to non-subscribers, you need to use Tivoli Software Distribution options set in an import file package definition.

#### **Example:**

```
(DATA
.
.
(NODES "tcaix01 tcaix02")
.
.
```

### **IMPORT**

Use this keyword to select Tivoli Software Distribution options not supported in the -o parameter of the teamcpak softdist command. The *filename* parameter is the name of a Tivoli Software Distribution import file package definition. You can generate an import file using the Tivoli Software Distribution user interface. If you use the IMPORT keyword, then instead of calling the standard Tivoli Software Distribution packaging command the Tivoli Software Distribution tool will call wimprftp to get all of the Tivoli Software Distribution configuration options. Using the IMPORT keyword disables other options and causes errors if they are specified.

If you specify the **IMPORT** keyword, do not specify the **DISTRIBUTE**, **INSTALLPGM**, **LOGNODE**, or **LOGFILE** keywords.

If you use the **INCLUDE** option in an import file, it is overridden by the list of files provided to the **teamcpak** command.

**Example:**

```
(DATA
.
.
  (IMPORT importFile)
.
.)
```

**DISTRIBUTE**

Specify **FULL** to distribute all files or **CHANGED** to distribute only those changed since the last distribution. The default is **FULL**.

**Example:**

```
(DATA
.
.
  (DISTRIBUTE CHANGED)
.
.)
```

**INSTALLPGM**

Use this keyword to specify an installation script to be run during distribution on each node that receives files. Specify the full file path name of the script.

**Example:**

```
(DATA
.
.
  (INSTALLPGM /tivoli/fpTeamcAIX/tcinstl.ksh)
.
.)
```

**LOGNODE**

This keyword specifies the system on which the log file is located. The node name you specify must be a managed node. The default is the current build machine or a machine running **teamcpak**.

**Example:**

```
(DATA
.
.
  (LOGNODE tcaix04)
.
.)
```

**LOGFILE**

This keyword specifies the directory path and file name of the log file on the log node. Include this keyword only if you use the **LOGNODE** keyword. The default value for this keyword is **softdist.log**.

**Example:**

```
(DATA
.
.
  (LOGFILE /tmp/softdist.log)
.
.)
```

---

## Problem determination for the Tivoli Software Distribution tool

If you are having trouble distributing files using the Tivoli Software Distribution distribution tool, you can use the following tools or teamcpak options to determine what the problem is:

### Log file

Check the softdist.log file (or the file name you specified in the LOGFILE keyword) for error messages.

**Mail** Check Tivoli mail messages generated during the distribution.

### -k option

Run the teamcpak command with the -k option to keep the package file after the distribution has been run.

### -x option

Run the teamcpak command with the -x option to leave any distributed files on the target.

### Trace facility

Run teamcpak with the trace facility. Use this facility only under guidance of an IBM service representative. See "Using the trace facility" on page 176 for more information.

The following message displays when a Tivoli Software Distribution command fails during a distribution.

```
6022-303 Tivoli/Software Distribution %s command failed with return code: RC.
To correct problem use:
- package file parameters LOGNODE and LOGFILE to record Tivoli output,
- packaging option "-k" to keep Tivoli File Package and teamcpak log file
  or "-m" to ignore input errors,
- packaging option "-x" to not clean up files that are distributed,
- TeamConnection Trace facility (see TeamConnection Administration Guide)
- or Tivoli Trace facility (see Tivoli documentation)
```

---

## Sample package file

The following is an example of scripts and items required to automatically execute packaging, distribution, and installation of files in a AIX-based system.

- The **teamcpak** command syntax that will execute subcommands or scripts for the package, distribute, and install functions.

```
teamcpak -i -o "-a -n -t" softdist Client.lst
```

- The **Client.pkf** file you create containing keywords and parameters for distributing and packaging functions.

```
(DATA
(PACKAGEFORMAT softdist)
(TARGETROOT /user/local/teamc/images)
(MANAGER Distribi)
(NODES perlvr tcaix02)
(INSTALLPGM /tivoli/fpTeamcAIX/tcinstall.ksh)
(LOGNODE tcaix00)
(LOGFILE /tmp/fpTeamcAIX.log)
)
```

- The **Client.lst** file you create containing the list of files passed to **teamcpak**. The first line contains the package file by convention. The example also contains customized installation files (**tcinstall.ksh**), TeamConnection tar files, and an installation script (**tcinst.ksh**).

```
Client.lst;
/usr/teamc/tivoli/Client.pkf
/usr/teamc/tivoli/tcinstall.ksh
/tcinstall/v208/fullpak/aix4/tar/client.tar
/tcinstall/v208/fullpak/aix4/tar/msgen_us.tar
/tcinstall/v208/fullpak/tcinst.ksh
```

The following presents an example of a Tivoli installation script (**tcinstall.ksh**) that is copied to the target along with the tar files and the TeamConnection installation script (**tcinst.ksh**), then executed on the target.

```
#!/bin/ksh
# Clear existing log

INST_DIR=/usr/local/teamc
INST_TMP=${INST_DIR}/tcinstl.tmp
INST_OUT=${INST_DIR}/tcinstl.out
INST_ERR=${INST_DIR}/tcinstl.err
INST_LOG=${INST_DIR}/tcinstl.log
IMAGE_DIR=${INST_DIR}/images

rm -f $INST_ERR $INST_OUT $INST_LOG $INST_TMP >/dev/null 2>&1
mkdir -p ${IMAGE_DIR}

exec 1>${INST_ERR}
exec 2>&1
exec 3>${INST_LOG}

# Install TeamConnection using responsefile
print -u3 Starting TeamC installation at 'date'
print -u3 User id= 'id'
print -u3 Input: $*

# Set up installation environment
# - assumes bourne or korn shell
grep OS_ROOTDIR '/.profile'
if (($? != 0))
then
    print -u3 Updating /.profile
    exec 4>>/.profile
    cd /
    print -u3 'ObjectStore and TeamConnection settings'
    print -u4 'OS_ROOTDIR=/usr/lpp/ODI/OS4.0'
    print -u4 'export OS_ROOTDIR'
    print -u4 'PATH=$PATH:$OS_ROOTDIR/cset/bin'
    print -u4 'export PATH'
    print -u4 'LIBPATH=$LIBPATH:$OS_ROOTDIR/common/lib'
    print -u4 'export LIBPATH'
    . /.profile
else
    print -u3 /.profile already updated
fi

# Set up error logging
# - if *.warning is in file (preceded by spaces and tabs only
grep "^[ ]*\.warning" /etc/syslog.conf
if (($? != 0))
then
    print -u3 'Updating /etc/syslog.conf'
    touch /var/spool/syslog
    chmod 666 /var/spool/syslog
    exec 4>> /etc/syslog.conf
```

```

        print -u4 '*.warning /var/spool/syslog'
        stopsrc -s syslog
        startsrc -s syslog
    else
        print -u3 /etc/syslog.conf already updated
    fi

    # Update services file for tcocto family
    grep "tcocto" /etc/services
    if (($? != 0))
    then
        print -u3 Updating /etc/services
        exec 4>> /etc/services
        print -u4 'tcocto 8888/tcp'
    else
        print -u3 /etc/services already updated
    fi

    # Generate response file
    ###
    ### Change to use enviroment variables!!
    ###
    print -u5 '1'
    print -u5 '5'
    print -u5 '/usr/local/teamc/images'
    print -u5 '/usr/local/teamc'
    print -u5 '/usr/local/teamc/nls'
    print -u5 'en_US'
    print -u5 '/usr/local/teamc/X11'
    print -u5 ''
    print -u5 'i'

    # Run provided TeamC install script
    ls -laR ${IMAGE_DIR} >> ${INST_TMP} 2>&1
    cd ${IMAGE_DIR}
    ${IMAGE_DIR}/tcinst.ksh < ${IMAGE_DIR}/tcinst1.response
    if (($? != 0))
    then
        # Failed installation
        print -u3 TeamC installation failed
        exit 1
    else
        # Clean up installation directory after listing contents
        print -u3 We have successfully copied TeamC installation files
        print -u3 Installation directory contents:
        ls -laR ${INST_DIR} >> ${INST_TMP} 2>&1
    fi

    cd /
    # Remove installation stuff
    print -u3 TeamC cleaning up temporary installation directory
    rm -rf ${IMAGE_DIR}
    cat ${INST_TMP} >> ${INST_LOG}
    rm -rf ${INST_TMP}
    exit 0

# end of file

```



---

## Appendix A. Family administration commands

On OS/2 and Windows server platforms, you can use either the Family Administrator GUI or the command line to configure TeamConnection families. For AIX, Solaris, and HP-UX server platforms, you can use the command line to configure TeamConnection families. This appendix explains how to do these tasks using the various family administrator commands.

Doing these tasks from the command line sometimes requires extra steps and is more prone to error. For these reasons, we recommend that you use the GUI when you can. See “Part 3. Designing and creating your TeamConnection environment” on page 97 for instructions on using the GUI.

This appendix explains how to do the following tasks from a command line:

To do this task,	Go to this page.
Creating a family database	369
Creating an initial superuser for a family	370
Creating or modifying authority groups	371
Creating or modifying interest groups	372
Configuring component or release processes	373
Defining configurable field types	375
Creating or modifying configurable fields	378
Changing report formats	381
Configuring user exits	384

---

### Creating a family database

You can create a family database from a command line prompt using the `fhcirt` command, as follows:

```
fhcirt dbpath\familyname
```

Where:

- *dbpath* is the name of the drive and directory where you want to create the family database. This directory must exist before you issue the `fhcirt` command. If you do not include a full file path specification, TeamConnection creates the family database in the directory from which you issue the `fhcirt` command.
- *familyname* is the name of the family to create.

The following example creates a family database named `family1` in the directory `family1` on drive `d`:

```
fhcirt d:\family1\family1
```

This command creates a file called `family1.tcd` in `d:\family1`.

The following example creates a family database named `family2` in the current directory:

```
fhcirt family2
```

This command creates a file called family2.tcd in the current directory.

---

## Creating an initial superuser for a family

Before you can define users, components, and releases for a family, you need to create a user ID with superuser access to the family. From a command line, you can do this using the fhchdf command. Before you can use this command, you need to create the family database and make sure the environment variable TC\_DBPATH is set. You can issue this command only once for each family. After the initial superuser ID has been created, use the TeamConnection GUI or line commands to modify or create additional users. If the family database has a component called "root," then the fhchdf command will not execute.

```
fhchdf -create
-user Name
-login Name
-address Name
-family Name
[-name Text]
[-area Name]
[-password Name]
```

Where:

- -user *Name* is the TeamConnection user ID for the superuser. If you omit this parameter, it defaults to the value specified for the -login parameter. It is a good idea to give the superuser an ID that is readily identifiable as a superuser. A good way to do this is to preface the user ID with su\_, such as -user su\_john.

**Note:** This parameters is used only in single-user environments, such as OS/2.

- -login *Name* is the login ID for the superuser. This parameter is used in multiuser environments, such as AIX, HP-UX, and Windows NT, to identify the user account to which the TeamConnection superuser ID is assigned. Single-user environments, such as OS/2, do not define a separate login ID. If you omit the -user parameter, it defaults to the value specified for the -login parameter. It is a good idea to give the superuser an ID that is readily identifiable as a superuser. A good way to do this is to preface the user ID with su\_, such as -login su\_john.
- -address *Name* is the hostname of the family server from which the superuser will be authorized to access the family, such as -address tcserver.
- -family *Name* is the name of the family for which you are defining the superuser. The family must have already been created. An example is -family testfam.
- -name *Text* is the real name of the superuser, such as -name "John Smith". This attribute is optional.
- -area *Name* is the development area in which the superuser works, such as -area "User interface". This attribute is optional.
- -password *Name* is the password that must be used by the initial superuser. If a password is not created, then no one will have access to the database. A password is required only if you created the family with the PASSWORD\_ONLY or PASSWORD\_OR\_HOST level of security.

The following example creates a superuser ID for John Smith on the family server tcserver for the family named robot:

```
fhchdf -create -user su_john -login jsmith -address tcserver
        -family robot -name "John Smith" -area "User interface"
        -password f5asdfjk
```

---

## Creating or modifying authority groups

When a TeamConnection family is created, the authority table is primed with default information contained in the `authorit.ld` file. This section provides instructions for manually editing the `authorit.ld` file to create new authority groups or change information about existing authority groups. When you change the `authorit.ld` file, you must also reload the contents of the authority table in the TeamConnection database.

Instructions for using the Family Administrator GUI to create or modify authority groups are on page 129.

Before you create or modify authority groups, you should be familiar with the information in “Planning for user access to TeamConnection data” on page 127.

## Editing the `authorit.ld` file

To add new authority groups or to add actions to an existing authority group, edit the `authorit.ld` file. It is recommended that you keep the `authorit.ld` file in the family directory. If you want to maintain common authority group definitions for more than one family, however, you can store this file in a common directory; but you will need to specify the fully-qualified path name for the `authorit.ld` file when you load it using the `fhclauth` command.

Add entries to the file using the following format:

`AuthorityGroup|ActionName`

### **AuthorityGroup**

This is the name of an existing authority group or the name of a group that you are creating. The name can be 15 characters long; it cannot contain blanks, tabs, or vertical separators. For an existing authority group, type the name exactly as it appears in the database table. The default names provided by IBM use all lowercase characters.

### **ActionName**

This is the name of an existing TeamConnection action. Specify only one action per entry. You must type the name exactly as it appears in the database table. See the list of actions in “Appendix F. Authority and notification for TeamConnection actions” on page 457 for the correct spelling and capitalization. Certain actions cannot be included in an authority group. These actions are noted in the table found in “Appendix H. Worksheets” on page 479 .

## Reloading the authority table

Whenever you change the `authorit.ld` file, you must reload the contents of the authority table before your users can use the new and changed authority groups.

You can reload the authority table as often as necessary. We recommend that you stop the family server before you reload the authority table (see page “Stopping the servers” on page 95 for instructions).

To reload the authority table, issue the following command from the server machine. Before issuing this command, ensure that the `TC_FAMILY` environment variable is set to the correct family name.

```
fhclauth path\authorit.ld dbpath\familyname
```

Where:

- *path\authorit.ld* is the path name of the *authorit.ld* file. If you specify a fully-qualified path name, TeamConnection looks for the *authorit.ld* file in the path you specify. If you specify only **authorit.ld** with no directory path, TeamConnection looks for the file in the directory specified by the TC\_DBPATH environment variable.
- *dbpath\familyname* is the path name of your TeamConnection family.

To verify that the authority table loaded correctly, use the report command to generate a report on the authority table. For example, to verify that a new authority group named *general* was added to the table, type the following from a client machine on an OS/2 or TeamConnection command line:

```
teamc report -view authority -where "name='general'"
```

If the table loaded correctly, information about the *general* authority group appears. If the authority table did not load correctly, make the necessary changes to the *authorit.ld* file and run the *fhclauth* command again.

For more information about the report -view command, refer to the *Commands Reference*.

---

## Creating or modifying interest groups

This section provides instructions for manually editing the *interest.ld* file to create or modify interest groups. When you change the *interest.ld* file, you must also reload the contents of the interest table.

Instructions for using the Family Administrator GUI to create or modify interest groups are on page 135.

Before you create or modify interest groups, you should be familiar with the information in "Planning for user notification" on page 134.

## Editing the interest.ld file

To add new interest groups or to add actions to an existing interest group, edit the *interest.ld* file. It is recommended that you keep the *interest.ld* file in the family directory. If you want to maintain common interest group definitions for more than one family, however, you can store this file in a common directory; but you will need to specify the fully-qualified path name for the *interest.ld* file when you load it using the *fhclintr* command.

Add entries to the file using the following format:

```
InterestGroup|ActionName
```

### InterestGroup

This is the name of an existing interest group or the name of a group that you are creating. The name can be up to 15 characters; it cannot contain blanks, tabs, or vertical separators. For an existing interest group, type the name exactly as it appears in the database table. The default names provided by IBM use all lowercase characters.

### ActionName

This is the name of an existing TeamConnection action. Specify only one action per entry. You must type the name exactly as it appears in the database table. See the list of actions in “Appendix F. Authority and notification for TeamConnection actions” on page 457 for the correct spelling and capitalization. Certain actions cannot be included in an interest group. These actions are noted in the table found in “Appendix H. Worksheets” on page 479 .

## Reloading the interest table

Whenever you change the interest.ld file, you must reload the contents of the interest table before your users can use the new and changed interest groups.

You can reload the interest table as often as necessary. We recommend that you stop the family server before you reload the interest table (see page “Stopping the servers” on page 95 for instructions).

To reload the interest table, issue the following command from the server machine in the directory where the interest.ld file is stored. Before issuing this command, ensure that the TC\_FAMILY environment variable is set to the correct family name.

```
fhclintr path\interest.ld dbpath\familyname
```

Where:

- *path\interest.ld* is the path name of the interest.ld file. If you specify a fully-qualified path name, TeamConnection looks for the interest.ld file in the path you specify. If you specify only **interest.ld** with no directory path, TeamConnection looks for the file in the directory specified by the TC\_DBPATH environment variable.
- *dbpath\familyname* is the path name of your TeamConnection family.

To verify that the interest table loaded correctly, use the report command to generate a report on the interest table. For example, to verify that a new interest group named *general* was added to the table, type the following from a command line on a client machine:

```
teamc report -view interest -where "name='general'"
```

If the interest table loaded correctly, information about the general interest group appears. If the interest table did not load correctly, make the necessary changes to the interest.ld file and run the command again.

For more information about the report -view command, refer to the *Commands Reference*.

---

## Configuring component or release processes

This section provides instructions for manually editing the comproc.ld and relproc.ld files to configure processes. When you change the .ld files, you must also reload the contents of the configurable process tables.

Instructions for using the Family Administrator GUI to configure processes are on page 153.

Before you configure processes, you should be familiar with the information in “Chapter 14. Configuring family processes” on page 153.

## Editing the `comproc.ld` and `relproc.ld` files

Information about configurable processes for components is stored in the `comproc.ld` file. Information about configurable processes for releases is stored in the `relproc.ld` file. When the family is created, the configurable process tables are created, based on the settings in the `comproc.ld` and `relproc.ld` files. If you modify the configurable process tables after the family is created, edit the `.ld` files and then run the `fhclproc` command.

To add new processes or change existing processes, edit the `comproc.ld` file for component processes or the `relproc.ld` file for release processes. It is recommended that you keep the `comproc.ld` and `relproc.ld` files in the family directory. If you want to maintain common process definitions for more than one family, however, you can store these files in a common directory; but you will need to specify the fully-qualified path name for them when you load them using the `fhclproc` command.

Add entries to the file using the following format:

`ProcessName|SubprocessName`

### **ProcessName**

The name of the process you are creating. The name can be up to 15 characters in length; it cannot contain blanks, tabs, or vertical separators.

### **SubprocessName**

The name of a `TeamConnection` subprocess. You can specify only one of the following subprocesses for each entry. If you want to include more than one subprocess, you must have an entry for each subprocess. Type the name exactly as it appears in the database.

The following are the subprocesses for components:

- none
- `dsrDefect`
- `dsrFeature`
- `verifyDefect`
- `verifyFeature`

The following are the subprocesses for releases:

- none
- `approval`
- `fix`
- `driver`
- `test`
- `track`

See “Release processes” on page 116 for an explanation of these subprocesses.

## Reloading the configurable process tables

After you edit an `.ld` file, use the `fhclproc` command to reload the contents of the configurable component or release process tables with the changed values. Before

issuing this command, ensure that the TC\_FAMILY environment variable is set to the correct family name. The format of the fhclproc command when reloading the component process table is:

```
fhclproc path\comproc.ld dbpath\familyname c
```

The format of the fhclproc command when reloading the release process table is:

```
fhclproc path\relproc.ld dbpath\familyname r
```

Where:

- *path\comproc.ld* or *path\relproc.ld* is the path name of the process definition file. If you specify a fully-qualified path name, TeamConnection looks for the .ld file in the path you specify. If you specify only the file name with no directory path, TeamConnection looks for the file in the directory specified by the TC\_DBPATH environment variable.
- *dbpath\familyname* is the path name of your TeamConnection family.
- *c* indicates that you are reloading the component process table.
- *r* indicates that you are reloading the release process table.

To verify that the command successfully modified the tables, use the report command to generate a report. To do this, type one of the following commands from an OS/2 or TeamConnection command line:

```
teamc report -view Cfgcomproc
teamc report -view Cfgrelproc
```

If the table did not load correctly, make the necessary changes to the comproc.ld or relproc.ld file and run the command again.

For more information about the report -view command, refer to the *Commands Reference*.

---

## Defining configurable field types

This section provides instructions for manually editing the config.ld file to define configurable field types. When you change the config.ld file, you must also reload the contents of the config table.

Instructions for using the Family Administrator GUI to define field types are on page 142.

Before you define field types, you should be familiar with the information in “Defining configurable field types” on page 142.

It is recommended that you keep the config.ld file in the family directory. If you want to maintain common configurable field definitions for more than one family, however, you can store this file in a common directory; but you will need to specify the fully-qualified path name for it when you load it using the fhclcnfg command.

When adding entries to the file, follow the existing format of the file:

```
fieldType|value|default|0|0|description
```

Information about configurable field types is stored in the config table. After you modify the config table, you must reload it (see “Reloading the config table” on page 377 ).

The config table consists of the following information:

**Field type**

Identifies the types of configurable fields that are defined for your family. You specify one of these types when you configure a new field. You can create new types, and you can configure the acceptable values for each type. You must have at least one value for each type. The type field can have up to 15 characters, but it cannot contain blank spaces or tabs.

The following describes the configurable field types that are shipped by TeamConnection:

**priority** An indication of the timing or scheduling requirements for resolving a defect or feature.

**drivertype**  
The type of driver the defect or feature resolution should be included in.

**severity**  
An indication of how severe a defect is.

**answerAccept**  
The answer to a defect or feature that the owner uses when accepting it.

**answerReturn**  
The answer to a defect or feature that the owner uses when returning it.

**defectPrefix**  
A prefix indicating the type of defect. The prefix attribute can distinguish a defect from a feature when a user looks at information regarding both. Use unique prefixes for defects.

**featurePrefix**  
A prefix indicating the type of feature. The prefix attribute can distinguish a defect from a feature when a user looks at information regarding both. Use unique prefixes for features.

**phase** The current stage in development when the defect was discovered or when the defect was injected into the code.

**symptom**  
An indication of the problem.

**Note:** If the default configurable fields shipped by IBM are not installed, the priority, phase, and symptom types are not used.

**Value** This field represents the choices the user has for the configurable field. You can add choices to the default fields shipped by TeamConnection and to the fields created specifically for your family. The value can have up to 85 characters (single-byte or double-byte); but it cannot contain spaces or tabs.

**Note:** Because a user can abbreviate these values from the command line, you cannot define a value that can be an abbreviation of another value of the same type. For example, you cannot add a value of build to the phase type, because a value of building already exists. Also, if a value of 1 exists for the severity type, you cannot add a severity value of 12.



**Default**

This field indicates whether the defined name is used as the default when the user does not enter a value for the configuration type. Valid values are either yes or no, and only one name for each configuration type can have the default field set to yes. The config table shipped by TeamConnection does not have any of the values set as defaults.

**Value1 and Value2**

Keep these fields set to 0. They are reserved for future use.

**Description**

This field contains the description of each value. The description field cannot contain more than 63 characters, but it can be set to blank. The description with the defined values appears on the GUI window when the field is displayed.

The default configuration field types, along with their attributes, that IBM ships are listed starting on page 387.

## Reloading the config table

When you edit the config.ld file and change any values, you must reload the contents of the config table so that TeamConnection recognizes the changes.

You can reload the config table as often as necessary. It is recommended that you stop the family server before you reload the table (see page “Stopping the servers” on page 95 for instructions).

To reload the config table, issue the following command from the server machine. Before issuing this command, ensure that the TC\_FAMILY environment variable is set to the correct family name.

```
fhclcnfg path\config.ld dbpath\familyname
```

Where:

- *path\config.ld* is the path name of the configurable fields definition file. If you specify a fully-qualified path name, TeamConnection looks for the file in the path you specify. If you specify only the file name with no directory path, TeamConnection looks for the file in the directory specified by the TC\_DBPATH environment variable.
- *dbpath\familyname* is the path name of your TeamConnection family.

Changing values in the config table does not change any values that are already in the database for existing records.

To verify that the command successfully modified the config table, use the report command to generate a report. To do this, type the following from an OS/2 or TeamConnection command line:

```
teamc report -view config
```

If the config table did not load correctly, make the necessary changes to the config.ld file and run the command again.

For more information, about the report -view command, refer to the *Commands Reference*.

---

## Creating and modifying configurable fields

This section provides instructions for using the `chfield` command to create and modify configurable fields. Before you use this command, you should be familiar with the information in “Changing or creating configurable fields” on page 144. If you use the Family Administrator GUI to create configurable fields, use it also to modify them. Using the `chfield` command to modify a configurable field created with the Family Administrator GUI can cause unpredictable results.

Instructions for using the Family Administrator GUI to create and modify configurable fields is on page 142.

From the server machine, use the `chfield` command to:

- Change the properties of existing configurable fields (provided they were created using `chfield`)
- Create new configurable fields
- Copy configurable fields from another family
- Create report formats for new fields

Stop the family server before you use the `chfield` command to change configuration field information (see page “Stopping the servers” on page 95 for instructions).

The following is the syntax for the `chfield` command:

```
chfield -object name [-source name]
```

Where:

- *object name* is the object to configure. Valid values are Defect, Feature, Part, PartView, or User .



PartView is valid only in UNIX environments. These values are case-sensitive. Be sure to specify them exactly as shown here.

- *source name* is the name of a system-generated file containing configurable fields. Valid values are `default` or an existing family name. The default family is the value of the `TC_FAMILY` environment variable.

If you specify an existing family name that is not fully qualified, the directory path of the current TeamConnection family is used. This is the path specified in the `TC_DBPATH` environment variable in your `config.sys` file.

Type the following command to display the configurable fields for the specified object (defect, feature, part, or user) in the current family:

```
chfield object
```

When you issue the `chfield` command, system files residing in the `cfgfield` subdirectory of the directory where the family database is installed are either generated or updated. The new fields do not take effect until the server is started again. The files are the following:

For the defect object:	<ul style="list-style-type: none"> <li>• cfgfield\Defect.tbl</li> <li>• cfgfield\Defect.fmt</li> </ul>
For the feature object:	<ul style="list-style-type: none"> <li>• cfgfield\Feature.tbl</li> <li>• cfgfield\Feature.fmt</li> </ul>
For the part object:	<ul style="list-style-type: none"> <li>• cfgfield\Part.tbl</li> <li>• cfgfield\Part.fmt</li> </ul>
For the user object:	<ul style="list-style-type: none"> <li>• cfgfield\User.tbl</li> <li>• cfgfield\User.fmt</li> </ul>

**Notes:**

1. Do *not* manually modify these files; the chfield program does that for you. Manually modifying the files can cause incorrect values to appear in your configurable fields.
2. The Part.tbl and User.tbl do not exist until they are generated by the chfield program.

## Creating configurable fields

Do the following to create a configurable field:

1. Stop the family server (see page “Stopping the servers” on page 95 for instructions).
2. Type the chfield command, specifying the object to which you are adding the field (for example, chfield -object defect). You are prompted to select an option.
3. Select option 1, Create a configurable field in the report and table file. You are prompted for the following information. You must type information at each prompt.

**Enter DB Column Name**

This is the name of the configurable field for the object.

**Is field active**

This flag indicates whether the field is active. A value of yes means the field is active; no means it is inactive. An inactive field is ignored, and its location in the database is replaced by the next field entry in the .tbl file.

**Enter CMD Attribute**

This name is used as the command attribute on the command line. If you enter **publImpact** at this prompt, for example, then the attribute for the PublImpact field on the defect command will be -publImpact.

**Enter Field Label**

This name appears in the user interface window as the field name.

**Enter Title**

This name appears in the header title for the object window—for example, PublicationImpact. Spaces are not allowed in the title. If this name is not provided, the configurable field is not displayed in the object window.

**Is it an attribute for "create/open" action**

This flag shows whether the field is one of the attributes within the user create, defect open, or feature open, actions. Valid values are yes and no.

**Is it a required field for "create/open" action**

This flag indicates whether the field is required for the create or open actions. Valid values are yes and no.

**Enter the field type**

This value matches one of the configurable field types defined in the config.ld file. See page 375 for a listing of the configuration field types that are shipped by TeamConnection. By specifying a type, you limit the acceptable values that a user can type in the field.

**Enter the width (in bytes) available for formatting**

The maximum allowed width of the value that can be entered in the field. This can be a number of 1 to 25.

When you exit the display, the file is changed, but the database is not altered until the server is stopped and then restarted.

Figure 67 shows an example of the display after typing information for creating a new field named developer.

```

Limit 15 characters per entry.

Enter DB Column Name :
developer
Is field active [Y/N] :
Enter CMD Attribute :
developer
Enter Field Label :
Developer:
Enter Title :
Developer
Is it an attribute for "create/open" action [Y/N] :
Y
Is it a required field for "create/open" action [Y/N] :
N
Enter the field type :
developer
Enter width (in bytes) available for formatting. Acceptable range of values: 1 to 25
10

```

*Figure 67. The chfield display with configurable field information filled in*

## Creating configurable fields by copying from another family

You can copy configurable field information from an existing family. To do this, do the following:

1. Stop the family server (see page "Stopping the servers" on page 95 for instructions).
2. If the source family is on a different machine than the target family, copy the .fmt and .tbl system files for each object that you want to copy from the source to the target machine.
3. Type the following command to create the fields for the specified object:

```
chfield -object objectName -source TargetFamilyName
```

4. Restart the family server so that TeamConnection recognizes the new fields.

To see the current field information, use the appropriate object command and the `-configInfo` action flag. For example, to see the field information for the user table in family rdev, type the following command:

```
teamc user -configInfo -family rdev
```

## Updating configurable fields

Do the following to update a configurable field:

1. Stop the family server (see page “Stopping the servers” on page 95 for instructions).
2. Type the `chfield` command, specifying the object in which you are updating the field (for example, `chfield -object defect`). You are prompted to select an option.
3. Select option 2, Update a configurable field in the report and table file.  
The current information for that object is displayed.
4. Type the DB column name you are updating.
5. For each prompt, either change the value or press Enter to keep the same value. Page 379 describes each prompt.
6. Restart the family server.

---

## Changing report formats

This section explains how you can manually change the position of report fields on the reports TeamConnection generates for the user, defect, feature, partFullView, and partView objects.

Instructions for using the Family Administrator GUI to change the reports are on page 147.

You can use the system editor to edit the following files. Before you change the report formats, you might want to make backup copies of these files.

- `cfgfield\Defect.fmt`
- `cfgfield\Feature.fmt`
- `cfgfield\Part.fmt`
- `chgfield\Partview.fmt`
- `cfgfield\User.fmt`

Each `.fmt` file is divided into five sections, separated by colons. The sections are:

- `StanzaViewFormat`
- `StanzaViewColumn`
- `TableViewFormat`
- `TableViewColumn`
- `TableViewHeader`

The column sections describe the column name of each of the labels specified in the format sections. The header section specifies how the columns appear in the table format.

The format sections specify the layout of the report. For example, a format specification of %3\$-25.25s indicates the following:

- %** Start of format specification.
- 3** The sequence number of the field that is generated by TeamConnection. The dollar sign must appear after the sequence number.
- The output is left-justified. If you do not include this character, the output is right-justified.
- 25** The minimum number of characters (bytes) of output.
- .25** The maximum number of characters (bytes) printed for all or part of the output field, or minimum number of digits printed for integer values.  
  
If you do not want the field displayed, type 0.0. For example, you have three sequence fields: 1, 2, and 3. If you do not want sequence 2 displayed, you type:  
%1\$-4.4s %2\$-0.0s %3\$-15.15s
- s** Type of data:
  - s for strings
  - ld for integers

You can specify only a data type of s for configurable fields. Use ld to display existing values, such as defect age.

You can also change or delete the format specification. Before you change a format specification, be aware of the following:

- A format specification in the stanza view does not have to match the format specification for the same field in the table view.
- Information in a stanza report appears in columns. When you specify the identical minimum and maximum number of characters for all fields appearing in a column, the report columns are left-justified. For example, Figure 68 on page 383 shows all the fields in the first column defined as 25.25.
- When you change a format specification in the table view, adjust the matching heading length in the table view header section. Otherwise, information will not appear correctly under the headings when users display the table.

Figure 68 on page 383 shows a sample report format for the defect table after configurable fields have been added. The changes are noted in bold font and are described following the figure.

```

# StanzaViewFormat
prefix      %1$s
name        %2$s
reference    %21$s
abstract     %9$s
duplicate    %13$s

state       %6$-25.25s  priority    %28$-20.20s
severity    %8$-25.25s  target      %29$-20.20s
age         %10$d

compName     %3$-25.25s  answer      %7$-20.20s
release      %4$-25.25s  symptom     %25$-20.20s
envName      %11$-25.25s phaseFound   %26$-20.20s
level        %12$-25.25s phaseInject  %27$-20.20s

addDate      %15$-25.25s assignDate   %16$-20.20s
lastUpdate   %14$-25.25s responseDate %17$-20.20s
endDate      %18$-25.25s

ownerLogin   %5$-25.25s  originLogin  %22$-20.20s
ownerName    %19$-25.25s originName    %23$-20.20s
ownerArea    %20$-25.25s originArea    %24$-20.20s

developer    %30$-25.25s

:
# StanzaViewColumn
prefix,name,reference,abstract,duplicate,state,priority,severity,target,age,compName,answer,
releaseName,symptom,envName,phaseFound,levelName,phaseInject,addDate,assignDate,lastUpdate,
responseDate,endDate,ownerLogin,originLogin,ownerName,originName,ownerArea,originArea,developer
:
# TableViewFormat
%1$-4.4s %2$-15.15s %3$-15.15s %6$-8.8s %22$-8.8s %5$-8.8s %8$-3.3s %10$-3.3ld %28$-4.4s %9$-55
.55s %4$-0.0s %7$-0.0s %11$-0.0s %12$-0.0s %13$-0.0s %14$-0.0s %15$-0.0s %16$-0.0s %17$-0.0s %1
8$-0.0s %19$-0.0s %20$-0.0s %21$-0.0s %23$-0.0s %24$-0.0s %25$-0.0s %26$-0.0s %27$-0.0s %30$-9.
9s
:
# TableViewColumn
prefix,name,compName,state,originLogin,ownerLogin,severity,age,priority,abstract,developer
:
# TableViewHeader
pref name      compName      state      originLo ownerLog sev age prio abstract developer
-----
:

```

Figure 68. Sample report format after adding configurable fields

In Figure 68, the format of the shipped defect report was modified as follows:

- Added a new label, `developer`, at the end of the `StanzaViewFormat` section, and the format specification `%30$-25.25s`
- Added the column name, `developer`, as the last entry in the `StanzaViewColumn` section

**Note:** When you edit the `StanzaViewColumn`, you must maintain a continuous line of text. Control characters are ignored and appear as output in the report.

- Added `%30$-9.9s` in the corresponding position for the `developer` entry in the `TableViewFormat` section
- Added the column name `developer` in the `TableViewColumn` section
- Added a new label, `developer`, in the `TableViewHeader` section and added the corresponding dashes in the next line

---

## Setting up user exits

This section provides instructions for manually updating the userExit file to add entries that call user-defined programs during the processing of TeamConnection actions.

Instructions for using the Family Administrator GUI to set up user exits are on page 162.

Before you edit the userExit file, you should be familiar with the information in “Chapter 15. Providing user exits” on page 155.

**Note:** The userExit file is copied to your family database directory from a file located in the language subdirectory of the nls\cfg directory path in the TeamConnection installation directory, for example, teamc\nls\cfg\enu. The version of the userExit file in this location contains comments that are not copied when the family is created using the Family Administrator GUI.

## Editing the userExit file

The userExit file has no defined actions until you add entries for the user exits that your organization will use. The entries you add specify the programs that you want started for specific TeamConnection actions. For each user exit, add an entry using the following format:

```
Action ExitID UProgram UParameter ENV=( ) #Comments
```

Use one or more blank spaces to separate each field in the entry. A line that begins with a # sign is a comment. You can have blank lines in the file.

The userExit file is located in the config subdirectory of the directory where your family’s database is installed.

A description of each field in the entry follows:

### Action

The name of the TeamConnection action that causes the user exit to start. You must type the name exactly as it appears in the database. See the list of actions in “Appendix H. Worksheets” on page 479 for the correct spelling and capitalization. For a list of actions that support user exits, see the User Exits page of the Settings notebook for your family.

**ExitID** Identifies when the user exit program is started during the course of the TeamConnection action. Valid values are 0, 1, 2, and 3. The value indicates that the user exit program does one of the following:

- 0** Starts at the beginning of the TeamConnection action, before any initialization or access checking takes place.
- 1** Starts after all TeamConnection checks are made and TeamConnection is ready to process the command.
- 2** Starts after the TeamConnection action is completed. At this point, the action has been submitted to TeamConnection, and all database or library updates have been committed.
- 3** Starts when a previous user exit with an exit ID of 0 or 1 is not



successful, or when the TeamConnection action is not successful. This exit ID allows the user exit program to clean up what the other user exit programs started.

**UEprogram**

The name of the user exit program. The program must exist in a directory defined in the PATH statement of your config.sys file (for OS/2 or Windows platforms).

**UEparameter**

A variable-length list of character string parameters provided to the user exit program.

**ENV=( )**

The customized parameter list for the user exit. See “Creating customized parameter lists” for more information on passing a customized parameter list to a user exit program.

**#Comments**

A comment about the user exit program. This field is optional.

TeamConnection does not recognize the updates to the userExit file until you stop and restart the TeamConnection server.

## Creating customized parameter lists

To create a customized parameter list for a user exit program, include the ENV=( ) field in the definition for the user exit in the userExit file. The ENV=( ) field consists of a comma-separated list of the parameters or configurable fields to be passed to the user exit program. To pass the component and release parameters of a PartAdd action to a user exit, for example, include the ENV=( ) field as follows:

```
ENV=(component,release)
```

See “Appendix C. User exit parameters” on page 393 for a list of parameters that can be passed to a user exit program for each action’s exit IDs.

To include a configurable field in a customized parameter list, identify it by its attribute name. See “Finding information about configurable fields” on page 390 for a guide to understanding how configurable fields are identified in different TeamConnection user interfaces.



---

## Appendix B. Configurable field types

This appendix describes the configurable field types shipped with TeamConnection. It also contains information that may help you determine how options that define configurable field types and configurable fields in the TeamConnection GUI, command line interface, and SQL interface correspond.

---

### Configurable field types

The following tables show the configuration table values under the following column headings:

**Field type**

Configuration field types that are supported by TeamConnection.

**Value** Values for the various configuration field types that are shipped with TeamConnection.

**Note:** Your TeamConnection family administrator can add names for each configuration field type.

**Description**

A description of each value shipped with TeamConnection.

**Note:** Your TeamConnection family administrator can add descriptions for fields.

There are no default values specified for the field types that IBM ships. However, your TeamConnection family administrator can set defaults for your family. For information on setting defaults, see “Defining configurable field types” on page 142.

#### The timing or scheduling requirements for resolving a defect or implementing a feature

Field Type	Value	Description
priority	mustfix	Defect or feature must be resolved in this release
priority	candidate	Defect or feature is a candidate if time permits
priority	deferred	Defect or feature deferred to next release
priority	easy	Defect or feature is easy to solve or implement
priority	moderate	Defect or feature is moderately difficult to resolve
priority	difficult	Defect or feature is difficult to solve or implement
priority	n/a	Priority does not apply to this defect or feature

### The type of driver

Field Type	Value	Description
drivertype	development	Development driver
drivertype	production	Production driver
drivertype	integration	Integration driver
drivertype	prototype	Prototype driver
drivertype	other	Other type of driver

### The severity of the problem that a defect was opened to resolve

Field Type	Value	Description
severity	1	Wrong results or failure; critical to program execution
severity	2	Wrong results; not critical to program execution
severity	3	Unexpected behavior
severity	4	Suggestion or enhancement request

### The type of defect or feature

Field Type	Value	Description
defectPrefix	c	Defect reported by a customer
defectPrefix	d	Defect reported by internal users
featurePrefix	s	Suggestion made by customer
featurePrefix	f	Feature requested by internal users

### The symptom of the problem a defect was opened to resolve

Field Type	Value	Description
symptom	incorrect_i/o	Incorrect or unexpected input or output
symptom	program_defect	Program defect
symptom	design_wrong	Original design is incorrect; redesign required
symptom	function_needed	Additional function is required
symptom	plans_incorrect	Plans need to be changed or enhanced
symptom	docs_incorrect	Documentation is incorrect
symptom	prog_suspended	Program suspended during normal operation
symptom	core_dump	Core dump occurred during normal operation
symptom	lost_data	Data loss occurred during normal operation
symptom	usability	Program or application is not usable as is
symptom	test_failed	Test failed
symptom	build_failed	Build, compile, or module integration failed
symptom	install_failed	Installation failed
symptom	obsolete_code	Remove obsolete code

Field Type	Value	Description
symptom	intgr_problem	Integration problems with other applications
symptom	performance	Performance problems; code needs to be optimized
symptom	reliability	Reliability problems; code needs more work
symptom	non-standard	Coding practices or program execution is non-standard
symptom	not_to_spec	Program or application does not function as specified

#### The development phase in progress when a defect was found or injected

Field Type	Value	Description
phase	design	Design Phase
phase	planning	Planning Phase
phase	strategy	Strategic Planning Phase
phase	prototyping	Prototyping Phase
phase	development	Development Phase
phase	documenting	Documentation or Publication Phase
phase	inspections	Inspection Phase
phase	maintenance	Maintenance Phase
phase	building	Building, Compiling or Module Integration Phase
phase	unit_test	Unit Test
phase	functional_test	Functional Test
phase	regression_test	Regression Test
phase	install_test	Installation Test
phase	config_test	Configuration Test
phase	integrate_test	Integration Test
phase	quality_test	Quality Assurance Test
phase	usability_test	Usability Test
phase	ship_test	Ship Test
phase	beta_test	Beta Test
phase	n/a	Not applicable to any particular phase

#### The reason a defect or feature is being accepted

Field Type	Value	Description
answerAccept	program_defect	The problem was due to a program error
answerAccept	docs_defect	Documentation needs to be changed
answerAccept	docs_change	Documentation needs to address new features
answerAccept	plans_change	Plans or schedules need to be changed

Field Type	Value	Description
answerAccept	new_function	New function will be added
answerAccept	redesign	Current function needs to be redesigned
answerAccept	fix_testcase	Testcase needs to be fixed
answerAccept	remove_code	Obsolete code needs to be removed
answerAccept	remove_support	Nonsupported functions need to be removed
answerAccept	comply_with	Coding practices and operation needs to comply with standards

#### The reason a defect or feature is being returned

Field Type	Value	Description
answerReturn	fixed	The problem is already fixed
answerReturn	future	Future releases or versions will address the defect or feature
answerReturn	duplicate	This is a duplicate of an existing defect or feature
answerReturn	usage_error	The problem is caused by incorrect usage
answerReturn	hardware_error	The problem is caused by a hardware error
answerReturn	info_needed	More information is required
answerReturn	limitation	This problem is a current limitation
answerReturn	suggestion	This problem is a suggestion, not an error
answerReturn	unrecreatable	The problem cannot be re-created
answerReturn	as_designed	The program works as designed
answerReturn	deviation	Code or documentation will deviate from the standards

## Finding information about configurable fields

Some of the options for defining configurable field types and configurable fields are identified differently according to the interface you use to work with them. You can view or change the definitions of configurable field types, for example, using any of the following interfaces:

- Editing the `config.ld` file directly
- The Family Administrator GUI
- Issuing **teamc report -view** queries from the command line

The following table shows how each part of a configurable field type definition is identified in each of these interfaces.

Table 23. Identifying configurable field type information

<code>config.ld</code>	Family Administrator GUI	Report -view
fieldType	Field Type	configType
value	Possible Values	name
default	Default	dflt

Table 23. Identifying configurable field type information (continued)

config.Id	Family Administrator GUI	Report -view
0	V1	value1
0	V2	value2
description	Description	description

You can view or change the definitions of configurable fields using any of the following interfaces:

- The Family Administrator GUI
- The **chfield** command
- Issuing **teamc object -configInfo** commands

The following table shows how each part of a configurable field definition is identified in each of these interfaces. For an explanation of where this information is used in the TeamConnection GUI, see “Creating and modifying configurable fields” on page 145.

Table 24. Identifying configurable field information

Family Administrator GUI	chfield	teamc object -configInfo
Configurable Fields	Enter DBColumn name	DBColumn Name
Active	Is field active?	
Required	Is it a required field for create/open action?	Required
Allow on Create/Open	Is it an attribute for create/open action?	Create
Field Type	Enter the field type	Field Type
Attribute	Enter CMD attribute	Attribute
Field Label	Enter field label	Field Label
Title	Enter title	Title





## Appendix C. User exit parameters

The following table shows the parameters passed to each user exit program defined for a specific TeamConnection action and ExitID. A description of the parameters follows the table on page 413.

**Note:** Parameters are not shown for exit ID 3. The parameters for exit ID 3 are the same as those passed to exit ID 0, with an additional parameter at the end to indicate the last user exit ID that has been executed successfully, for example, 0 or 1.

A parameter name followed by *not used* indicates that TeamConnection passes an empty string.

See “Chapter 15. Providing user exits” on page 155 for more information on user exits.

### Parameters passed to user exit programs

The figure that follows shows the parameters passed to each user exit program defined for a specific TeamConnection action and exit ID.

TeamConnection action	Exit ID	Parameters passed to the user exit program (see page 413 for definitions)
<b>Access</b>		
AccessCreate	0	NewOwner, component, authority, effectiveUserID, TeamcUserID, VerboseFlag
	1	NewOwner, component, authority, effectiveUserID, VerboseFlag
	2	NewOwner, component, authority, effectiveUserID, VerboseFlag
AccessDelete	0	OldOwner, component, authority, effectiveUserID, TeamcUserID, VerboseFlag
	1	OldOwner, component, authority, effectiveUserID, VerboseFlag
	2	OldOwner, component, authority, effectiveUserID, VerboseFlag
AccessRestrict	0	NewOwner, component, authority, effectiveUserID, TeamcUserID, VerboseFlag
	1	NewOwner, component, authority, effectiveUserID, VerboseFlag
	2	NewOwner, component, authority, effectiveUserID, VerboseFlag
ApprovalAbstain	0	release, WorkAreaName, ApproverName, effectiveUserID, TeamcUserID, VerboseFlag
	1	release, WorkAreaName, DefectOrFeatureName, ApproverName, workareaType, effectiveUserID, VerboseFlag
	2	release, WorkAreaName, DefectOrFeatureName, ApproverName, workareaType, effectiveUserID, VerboseFlag
ApprovalAccept	0	release, WorkAreaName, ApproverName, effectiveUserID, TeamcUserID, VerboseFlag

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
	1	release, WorkAreaName, DefectOrFeatureName, ApproverName, workareaType, effectiveUserID, VerboseFlag
	2	release, WorkAreaName, DefectOrFeatureName, ApproverName, workareaType, effectiveUserID, VerboseFlag
	0	release, WorkAreaName, OldOwner, NewOwner, effectiveUserID, TeamcUserID, VerboseFlag
ApprovalAssign	1	release, WorkAreaName, DefectOrFeatureName, OldOwner, NewOwner, workareaType, effectiveUserID, VerboseFlag
	2	release, WorkAreaName, DefectOrFeatureName, OldOwner, NewOwner, workareaType, effectiveUserID, VerboseFlag
	0	release, WorkAreaName, ApproverName, effectiveUserID, TeamcUserID, VerboseFlag
ApprovalCreate	1	release, WorkAreaName, DefectOrFeatureName, ApproverName, workareaType, effectiveUserID, VerboseFlag
	2	release, WorkAreaName, DefectOrFeatureName, ApproverName, workareaType, effectiveUserID, VerboseFlag
	0	release, WorkAreaName, ApproverName, effectiveUserID, TeamcUserID, VerboseFlag
ApprovalDelete	1	release, WorkAreaName, DefectOrFeatureName, ApproverName, workareaType, effectiveUserID, VerboseFlag
	2	release, WorkAreaName, DefectOrFeatureName, ApproverName, workareaType, effectiveUserID, VerboseFlag
	0	release, WorkAreaName, ApproverName, effectiveUserID, TeamcUserID, VerboseFlag
ApprovalReject	1	release, WorkAreaName, DefectOrFeatureName, ApproverName, workareaType, effectiveUserID, VerboseFlag
	2	release, WorkAreaName, DefectOrFeatureName, ApproverName, workareaType, effectiveUserID, VerboseFlag
	0	release, WorkAreaName, ApproverName, effectiveUserID, TeamcUserID, VerboseFlag
ApproverCreate	1	release, WorkAreaName, DefectOrFeatureName, ApproverName, workareaType, effectiveUserID, VerboseFlag
	2	release, WorkAreaName, DefectOrFeatureName, ApproverName, workareaType, effectiveUserID, VerboseFlag
	0	NewOwner, release, effectiveUserID, TeamcUserID, VerboseFlag
ApproverDelete	1	NewOwner, release, effectiveUserID, VerboseFlag
	2	NewOwner, release, effectiveUserID, VerboseFlag
	0	OldOwner, release, effectiveUserID, TeamcUserID, VerboseFlag
BuilderCreate	1	OldOwner, release, effectiveUserID, VerboseFlag
	2	OldOwner, release, effectiveUserID, VerboseFlag
	0	name, transmitFlag, temporaryfileonserver, release, condition, value, script, filetype, buildparameters, targetenvironment, timeout, processoroptions, effectiveUserID, TeamcUserID, VerboseFlag

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
	1	name, temporaryfileonserver, release, condition, value, script, filetype, buildparameters, targetenvironment, timeout, processoroptions, effectiveUserID, VerboseFlag
	2	name, temporaryfileonserver, release, condition, value, script, filetype, buildparameters, targetenvironment, timeout, processoroptions, effectiveUserID, VerboseFlag
	0	name, release, effectiveUserID, TeamcUserID, VerboseFlag
BuilderDelete	1	name, release, effectiveUserID, VerboseFlag
	2	name, release, effectiveUserID, VerboseFlag
	0	name, release, effectiveUserID, TeamcUserID, VerboseFlag
BuilderExtract	1	name, release, effectiveUserID, VerboseFlag
	2	name, temporaryfileonserver, release, condition, value, script, filetype, buildparameters, targetenvironment, timeout, processoroptions, effectiveUserID, VerboseFlag
	0	name, transmitFlag, temporaryfileonserver, release, condition, value, script, filetype, buildparameters, targetenvironment, timeout, processoroptions, effectiveUserID, TeamcUserID, VerboseFlag
BuilderModify	1	name, temporaryfileonserver, release, condition, value, script, filetype, buildparameters, targetenvironment, timeout, processoroptions, effectiveUserID, VerboseFlag
	2	name, temporaryfileonserver, release, condition, value, script, filetype, buildparameters, targetenvironment, timeout, processoroptions, effectiveUserID, VerboseFlag
	0	name, release, effectiveUserID, TeamcUserID, VerboseFlag
BuilderView	1	name, release, effectiveUserID, VerboseFlag
	2	name, release, effectiveUserID, VerboseFlag
	0	pathName, WorkAreaName, release, state, alternateversion, workareaType, typename, effectiveUserID, TeamcUserID, VerboseFlag
CollisionAccept	1	pathName, WorkAreaName, release, typename, effectiveUserID, VerboseFlag
	2	pathName, WorkAreaName, release, typename, effectiveUserID, VerboseFlag
	0	pathName, WorkAreaName, release, state, alternateversion, workareaType, typename, effectiveUserID, TeamcUserID, VerboseFlag
CollisionReconc	1	pathName, WorkAreaName, release, typename, effectiveUserID, VerboseFlag
	2	pathName, WorkAreaName, release, typename, effectiveUserID, VerboseFlag
	0	pathName, WorkAreaName, release, state, alternateversion, workareaType, typename, effectiveUserID, TeamcUserID, VerboseFlag
CollisionReject	1	pathName, WorkAreaName, release, state, alternateversion, workareaType, typename, effectiveUserID, TeamcUserID, VerboseFlag
	2	pathName, WorkAreaName, release, state, alternateversion, workareaType, typename, effectiveUserID, TeamcUserID, VerboseFlag
	0	pathName, WorkAreaName, release, state, alternateversion, workareaType, typename, effectiveUserID, TeamcUserID, VerboseFlag

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
	1	pathName, WorkAreaName, release, typename, effectiveUserID, VerboseFlag
	2	pathName, WorkAreaName, release, typename, effectiveUserID, VerboseFlag
	0	component, parentcomponent, owner, componentprocess, description, effectiveUserID, TeamcUserID, VerboseFlag
CompCreate	1	newcomponent, parentcomponent, owner, newcomponentprocess, description, effectiveUserID, VerboseFlag
	2	newcomponent, parentcomponent, owner, newcomponentprocess, description, effectiveUserID, VerboseFlag
	0	component, effectiveUserID, TeamcUserID, VerboseFlag
CompDelete	1	component, effectiveUserID, VerboseFlag
	2	component, effectiveUserID, VerboseFlag
	0	component, parentcomponent, effectiveUserID, TeamcUserID, VerboseFlag
CompLink	1	component, parentcomponent, effectiveUserID, VerboseFlag
	2	component, parentcomponent, effectiveUserID, VerboseFlag
	0	component, newcomponent, NewOwner, newdescription, newcomponentprocess, effectiveUserID, TeamcUserID, VerboseFlag
CompModify	1	component, newcomponent, OldOwner, NewOwner, olddescription, newdescription, oldcomponentprocess, newcomponentprocess, dateoflastupdate, effectiveUserID, VerboseFlag
	2	name, newcomponent, NewOwner, description, process, effectiveUserID, VerboseFlag
	0	component, parentcomponent, effectiveUserID, TeamcUserID, VerboseFlag
CompRecreate	1	component, parentcomponent, olddropDate, effectiveUserID, VerboseFlag
	2	component, parentcomponent, olddropDate, effectiveUserID, VerboseFlag
	0	component, parentcomponent, effectiveUserID, TeamcUserID, VerboseFlag
CompUnlink	1	component, parentcomponent, effectiveUserID, VerboseFlag
	2	component, parentcomponent, effectiveUserID, VerboseFlag
	0	component, displaytype, effectiveUserID, TeamcUserID, VerboseFlag
CompView	1	component, displaytype, effectiveUserID, VerboseFlag
	2	component, displaytype, effectiveUserID, VerboseFlag
	0	release, primeworkareaname, secondworkareaname, effectiveUserID, TeamcUserID, VerboseFlag
CoreqCreate	1	release, primeworkareaname, secondworkareaname, effectiveUserID, VerboseFlag

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
CoreqDelete	2	release, primeworkareaname, secondworkareaname, effectiveUserID, VerboseFlag
	0	release, WorkAreaName, effectiveUserID, TeamcUserID, VerboseFlag
	1	release, WorkAreaName, effectiveUserID, VerboseFlag
DefectAccept	2	release, WorkAreaName, effectiveUserID, VerboseFlag
	0	defectname, originaldefectname, answer, remarks, effectiveUserID, TeamcUserID, VerboseFlag
	1	defectname, answer, remarks, effectiveUserID, VerboseFlag
DefectAssign	2	defectname, originaldefectname, answer, remarks, effectiveUserID, VerboseFlag
	0	defectname, newcomponent, NewOwner, remarks, effectiveUserID, TeamcUserID, VerboseFlag
	1	defectname, newcomponent, NewOwner, remarks, effectiveUserID, VerboseFlag
DefectCancel	2	defectname, newcomponent, NewOwner, remarks, effectiveUserID, VerboseFlag
	0	defectname, originaldefectname, answer, remarks, effectiveUserID, TeamcUserID, VerboseFlag
	1	defectname, remarks, effectiveUserID, VerboseFlag
DefectComment	2	defectname, originaldefectname, answer, remarks, effectiveUserID, VerboseFlag
	0	defectname, remarks, effectiveUserID, TeamcUserID, VerboseFlag
	1	defectname, remarks, effectiveUserID, VerboseFlag
DefectDesign	2	defectname, remarks, effectiveUserID, VerboseFlag
	0	defectname, originaldefectname, answer, remarks, effectiveUserID, TeamcUserID, VerboseFlag
	1	defectname, remarks, effectiveUserID, VerboseFlag
DefectModify	2	defectname, originaldefectname, answer, remarks, effectiveUserID, VerboseFlag
	0	defectname, newdefectname, severity, environmentname, prefix, reference, drivename, abstract, originator, answer, remarks, release, configFields, effectiveUserID, TeamcUserID, VerboseFlag
	1	defectname, newdefectname, severity, environmentname, prefix, reference, drivename, abstract, originator, answer, remarks, release, configFields, dateoflastupdate, effectiveUserID, VerboseFlag
DefectOpen	2	defectname, newdefectname, severity, environmentname, prefix, reference, drivename, abstract, originator, answer, remarks, release, configFields, effectiveUserID, VerboseFlag
	0	component, prefix, severity, reference, environmentname, remarks, drivename, abstract, release, configFields, defectname, effectiveUserID, TeamcUserID, VerboseFlag

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
	1	component, prefix, severity, reference, environmentname, remarks, drivename, abstract, release, configFields, defectname, effectiveUserID, VerboseFlag
	2	component, prefix, severity, reference, environmentname, remarks, drivename, abstract, release, configFields, defectname, effectiveUserID, VerboseFlag
	0	defectname, originaldefectname, answer, remarks, effectiveUserID, TeamcUserID, VerboseFlag
DefectReopen	1	defectname, remarks, effectiveUserID, VerboseFlag
	2	defectname, originaldefectname, answer, remarks, effectiveUserID, VerboseFlag
	0	defectname, originaldefectname, answer, remarks, effectiveUserID, TeamcUserID, VerboseFlag
DefectReturn	1	defectname, originaldefectname, answer, remarks, effectiveUserID, VerboseFlag
	2	defectname, originaldefectname, answer, remarks, effectiveUserID, VerboseFlag
	0	defectname, originaldefectname, answer, remarks, effectiveUserID, TeamcUserID, VerboseFlag
DefectReview	1	defectname, originaldefectname, answer, remarks, effectiveUserID, VerboseFlag
	2	defectname, originaldefectname, answer, remarks, effectiveUserID, VerboseFlag
	0	defectname, originaldefectname, answer, remarks, effectiveUserID, TeamcUserID, VerboseFlag
DefectSize	1	defectname, remarks, effectiveUserID, VerboseFlag
	2	defectname, originaldefectname, answer, remarks, effectiveUserID, VerboseFlag
	0	defectname, originaldefectname, answer, remarks, effectiveUserID, TeamcUserID, VerboseFlag
DefectVerify	1	defectname, remarks, effectiveUserID, VerboseFlag
	2	defectname, originaldefectname, answer, remarks, effectiveUserID, VerboseFlag
	0	defectname, originaldefectname, answer, remarks, effectiveUserID, TeamcUserID, VerboseFlag
DefectView	1	defectname, remarks, effectiveUserID, VerboseFlag
	2	defectname, originaldefectname, answer, remarks, effectiveUserID, VerboseFlag
	0	defectname, displaytype, effectiveUserID, TeamcUserID, VerboseFlag
DriverAssign	1	defectname, displaytype, effectiveUserID, VerboseFlag
	2	defectname, displaytype, effectiveUserID, VerboseFlag
	0	drivename, release, NewOwner, effectiveUserID, TeamcUserID, VerboseFlag
DriverCheck	1	drivename, release, NewOwner, driverstate, drivertype, effectiveUserID, VerboseFlag
	2	drivename, release, NewOwner, driverstate, drivertype, effectiveUserID, VerboseFlag
	0	drivename, release, longFlag, effectiveUserID, TeamcUserID, VerboseFlag
	1	drivename, release, longFlag, driverstate, drivertype, effectiveUserID, VerboseFlag

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
DriverCommit	2	drivename, release, longFlag, driverstate, drivertype, effectiveUserID, VerboseFlag
	0	drivename, release, forceFlag, effectiveUserID, TeamcUserID, VerboseFlag
	1	drivename, release, drivertype, effectiveUserID, VerboseFlag
DriverComplete	2	drivename, release, drivertype, effectiveUserID, VerboseFlag
	0	drivename, release, effectiveUserID, TeamcUserID, VerboseFlag
	1	drivename, release, drivertype, effectiveUserID, VerboseFlag
DriverCreate	2	drivename, release, drivertype, effectiveUserID, VerboseFlag
	0	drivename, release, drivertype, effectiveUserID, TeamcUserID, VerboseFlag
	1	drivename, release, drivertype, effectiveUserID, VerboseFlag
DriverDelete	2	drivename, release, drivertype, effectiveUserID, VerboseFlag
	0	drivename, release, effectiveUserID, TeamcUserID, VerboseFlag
	1	drivename, release, driverstate, drivertype, effectiveUserID, VerboseFlag
DriverExtract	2	drivename, release, driverstate, drivertype, effectiveUserID, VerboseFlag
	0	drivename, release, root, nokeysFlag, ExtractType, fmask, dmask, crlfFlag, effectiveUserID, TeamcUserID, VerboseFlag
	1	drivename, release, root, nokeysFlag, ExtractType, fmask, dmask, crlfFlag, driverstate, drivertype, effectiveUserID, VerboseFlag
DriverFreeze	2	drivename, release, root, nokeysFlag, ExtractType, fmask, dmask, crlfFlag, driverstate, drivertype, effectiveUserID, VerboseFlag
	0	drivename, release, effectiveUserID, TeamcUserID, VerboseFlag
	1	drivename, release, driverstate, drivertype, effectiveUserID, VerboseFlag
DriverModify	2	drivename, release, driverstate, drivertype, effectiveUserID, VerboseFlag
	0	drivename, newdrivename, release, newdrivertype, effectiveUserID, TeamcUserID, VerboseFlag
	1	drivename, newdrivename, release, oldtype, newtype, driverstate, dateoflastupdate, effectiveUserID, VerboseFlag
DriverRefresh	2	drivename, newdrivename, release, oldtype, newtype, driverstate, effectiveUserID, VerboseFlag
	0	drivename, release, effectiveUserID, TeamcUserID, VerboseFlag
	1	drivename, release, driverstate, drivertype, effectiveUserID, VerboseFlag
DriverRestrict	2	drivename, release, driverstate, drivertype, effectiveUserID, VerboseFlag
	0	drivename, release, effectiveUserID, TeamcUserID, VerboseFlag
	1	drivename, release, drivertype, effectiveUserID, VerboseFlag

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
DriverView	2	drivername, release, drivertype, effectiveUserID, VerboseFlag
	0	drivername, release, displaytype, effectiveUserID, TeamcUserID, VerboseFlag
	1	drivername, release, displaytype, driverstate, drivertype, effectiveUserID, VerboseFlag
MemberCreate	2	drivername, release, displaytype, driverstate, drivertype, effectiveUserID, VerboseFlag
	0	drivername, release, WorkAreaName, forceFlag, effectiveUserID, TeamcUserID, VerboseFlag
	1	drivername, release, WorkAreaName, DefectOrFeatureName, workareastate, workareaType, effectiveUserID, VerboseFlag
MemberDelete	2	drivername, release, WorkAreaName, DefectOrFeatureName, workareastate, workareaType, effectiveUserID, VerboseFlag
	0	drivername, release, numberofworkareas, effectiveUserID, TeamcUserID, VerboseFlag
	1	drivername, release, effectiveUserID, VerboseFlag
EnvCreate	2	drivername, release, effectiveUserID, VerboseFlag
	0	environmentname, release, testersname, effectiveUserID, TeamcUserID, VerboseFlag
	1	environmentname, release, testersname, effectiveUserID, VerboseFlag
EnvDelete	2	environmentname, release, testersname, effectiveUserID, VerboseFlag
	0	environmentname, release, effectiveUserID, TeamcUserID, VerboseFlag
	1	environmentname, release, effectiveUserID, VerboseFlag
EnvModify	2	environmentname, release, effectiveUserID, VerboseFlag
	0	environmentname, release, newtestersname, effectiveUserID, TeamcUserID, VerboseFlag
	1	environmentname, release, newtestersname, effectiveUserID, VerboseFlag
FeatureAccept	2	environmentname, release, newtestersname, effectiveUserID, VerboseFlag
	0	featurename, originalfeaturename, remarks, StandardFields, configFields, effectiveUserID, TeamcUserID, VerboseFlag
	1	featurename, remarks, effectiveUserID, VerboseFlag
FeatureAssign	2	featurename, originalfeaturename, remarks, effectiveUserID, VerboseFlag
	0	featurename, newcomponent, NewOwner, remarks, effectiveUserID, TeamcUserID, VerboseFlag
	1	featurename, newcomponent, NewOwner, remarks, effectiveUserID, VerboseFlag



<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
FeatureCancel	2	featurename, newcomponent, NewOwner, remarks, effectiveUserID, VerboseFlag
	0	featurename, originalfeaturename, remarks, StandardFields, configFields, effectiveUserID, TeamcUserID, VerboseFlag
	1	featurename, remarks, effectiveUserID, VerboseFlag
FeatureComment	2	featurename, originalfeaturename, remarks, effectiveUserID, VerboseFlag
	0	featurename, remarks, effectiveUserID, TeamcUserID, VerboseFlag
	1	featurename, remarks, effectiveUserID, VerboseFlag
FeatureDesign	2	featurename, remarks, effectiveUserID, VerboseFlag
	0	featurename, originalfeaturename, remarks, StandardFields, configFields, effectiveUserID, TeamcUserID, VerboseFlag
	1	featurename, remarks, effectiveUserID, VerboseFlag
FeatureModify	2	featurename, originalfeaturename, remarks, effectiveUserID, VerboseFlag
	0	featurename, newfeaturename, prefix, reference, abstract, originator, remarks, configFields, effectiveUserID, TeamcUserID, VerboseFlag
	1	featurename, newfeaturename, prefix, reference, abstract, originator, remarks, configFields, dateoflastupdate, effectiveUserID, VerboseFlag
FeatureOpen	2	featurename, newfeaturename, prefix, reference, abstract, originator, remarks, configFields, effectiveUserID, VerboseFlag
	0	component, prefix, reference, remarks, abstract, configFields, featurename, effectiveUserID, TeamcUserID, VerboseFlag
	1	component, prefix, reference, remarks, abstract, configFields, featurename, effectiveUserID, VerboseFlag
FeatureReopen	2	component, prefix, reference, remarks, abstract, configFields, featurename, effectiveUserID, VerboseFlag
	0	featurename, originalfeaturename, remarks, StandardFields, configFields, effectiveUserID, TeamcUserID, VerboseFlag
	1	featurename, remarks, effectiveUserID, VerboseFlag
FeatureReturn	2	featurename, originalfeaturename, remarks, effectiveUserID, VerboseFlag
	0	featurename, originalfeaturename, remarks, StandardFields, configFields, effectiveUserID, TeamcUserID, VerboseFlag
	1	featurename, originalfeaturename, remarks, effectiveUserID, VerboseFlag
FeatureReview	2	featurename, originalfeaturename, remarks, effectiveUserID, VerboseFlag
	0	featurename, originalfeaturename, remarks, StandardFields, configFields, effectiveUserID, TeamcUserID, VerboseFlag
	1	featurename, remarks, effectiveUserID, VerboseFlag

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
FeatureSize	2	featurename, originalfeaturename, remarks, effectiveUserID, VerboseFlag
	0	featurename, originalfeaturename, remarks, StandardFields, configFields, effectiveUserID, TeamcUserID, VerboseFlag
	1	featurename, remarks, effectiveUserID, VerboseFlag
FeatureVerify	2	featurename, originalfeaturename, remarks, effectiveUserID, VerboseFlag
	0	featurename, originalfeaturename, remarks, StandardFields, configFields, effectiveUserID, TeamcUserID, VerboseFlag
	1	featurename, remarks, effectiveUserID, VerboseFlag
FeatureView	2	featurename, originalfeaturename, remarks, effectiveUserID, VerboseFlag
	0	featurename, displaytype, effectiveUserID, TeamcUserID, VerboseFlag
	1	featurename, displaytype, effectiveUserID, VerboseFlag
FixActive	2	featurename, displaytype, effectiveUserID, VerboseFlag
	0	WorkAreaName, release, component, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, DefectOrFeatureName, release, component, type, effectiveUserID, VerboseFlag
FixAssign	2	WorkAreaName, DefectOrFeatureName, release, component, type, effectiveUserID, VerboseFlag
	0	WorkAreaName, release, component, NewOwner, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, release, component, NewOwner, type, effectiveUserID, VerboseFlag
FixComplete	2	WorkAreaName, release, component, NewOwner, type, effectiveUserID, VerboseFlag
	0	WorkAreaName, release, component, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, DefectOrFeatureName, release, component, type, effectiveUserID, VerboseFlag
FixCreate	2	WorkAreaName, DefectOrFeatureName, release, component, type, effectiveUserID, VerboseFlag
	0	WorkAreaName, release, component, NewOwner, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, DefectOrFeatureName, release, component, NewOwner, type, effectiveUserID, VerboseFlag
FixDelete	2	WorkAreaName, DefectOrFeatureName, release, component, NewOwner, type, effectiveUserID, VerboseFlag
	0	WorkAreaName, release, component, effectiveUserID, TeamcUserID, VerboseFlag

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
	1	WorkAreaName, release, component, type, effectiveUserID, VerboseFlag
	2	WorkAreaName, release, component, type, effectiveUserID, VerboseFlag
	0	NewOwner, login@hostname, effectiveUserID, TeamcUserID, VerboseFlag
HostCreate	1	NewOwner, login@hostname, effectiveUserID, VerboseFlag
	2	NewOwner, login@hostname, effectiveUserID, VerboseFlag
	0	OldOwner, login@hostname, effectiveUserID, TeamcUserID, VerboseFlag
HostDelete	1	OldOwner, login@hostname, effectiveUserID, VerboseFlag
	2	OldOwner, login@hostname, effectiveUserID, VerboseFlag
	0	OldOwner, login@hostname, effectiveUserID, VerboseFlag
NotifyCreate	0	NewOwner, component, interestgroupname, effectiveUserID, TeamcUserID, VerboseFlag
	1	NewOwner, component, interestgroupname, effectiveUserID, VerboseFlag
	2	NewOwner, component, interestgroupname, effectiveUserID, VerboseFlag
NotifyDelete	0	OldOwner, component, interestgroupname, effectiveUserID, TeamcUserID, VerboseFlag
	1	OldOwner, component, interestgroupname, effectiveUserID, VerboseFlag
	2	OldOwner, component, interestgroupname, effectiveUserID, VerboseFlag
ParserCreate	0	description, release, parsercommand, paths, effectiveUserID, TeamcUserID, VerboseFlag
	1	description, release, parsercommand, paths, effectiveUserID, VerboseFlag
	2	description, release, parsercommand, paths, effectiveUserID, VerboseFlag
ParserDelete	0	description, release, effectiveUserID, TeamcUserID, VerboseFlag
	1	description, release, effectiveUserID, VerboseFlag
	2	description, release, effectiveUserID, VerboseFlag
ParserModify	0	description, release, parsercommand, paths, effectiveUserID, TeamcUserID, VerboseFlag
	1	description, release, parsercommand, paths, effectiveUserID, VerboseFlag
	2	description, release, parsercommand, paths, effectiveUserID, VerboseFlag
ParserView	0	description, release, effectiveUserID, TeamcUserID, VerboseFlag
	1	description, release, effectiveUserID, VerboseFlag
	2	description, release, effectiveUserID, VerboseFlag

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
PartAdd	0	partpathName, transmitFlag, filenameonclient, temporaryfileonserver, release, component, filetype, WorkAreaName, fMode, parentname, parsername, buildername, relationtoparent, buildparameters, parttype, parenttype, temporaryFlag, effectiveUserID, TeamcUserID, VerboseFlag
	1	partpathName, temporaryfileonserver, release, component, filetype, WorkAreaName, remarks, fMode, parentname, parsername, buildername, relationtoparent, buildparameters, parttype, parenttype, temporaryFlag, effectiveUserID, VerboseFlag
	2	partpathName, temporaryfileonserver, release, component, filetype, WorkAreaName, remarks, fMode, parentname, parsername, buildername, relationtoparent, buildparameters, parttype, parenttype, temporaryFlag, effectiveUserID, VerboseFlag
PartBuild	0	partpathName, release, WorkAreaName, buildmode, poolname, buildparameters, cancelFlag, detailfilename, clientportname, parttype, effectiveUserID, TeamcUserID, VerboseFlag
	1	partpathName, WorkAreaName, release, component, buildmode, poolname, buildparameters, cancelFlag, detailfilename, clienthostname, clientportname, parttype, effectiveUserID, VerboseFlag
	2	partpathName, WorkAreaName, release, component, buildmode, poolname, buildparameters, cancelFlag, detailfilename, clienthostname, clientportname, parttype, effectiveUserID, VerboseFlag
PartCheckIn	0	partpathName, transmitFlag, filenameonclient, temporaryfileonserver, release, forceFlag, WorkAreaName, commonFlag, filetype, parttype, retainlockFlag, effectiveUserID, TeamcUserID, VerboseFlag
	1	partpathName, temporaryfileonserver, release, component, forceFlag, WorkAreaName, remarks, commonreleases, filetype, parttype, retainlockFlag, configFields, effectiveUserID, VerboseFlag
	2	partpathName, temporaryfileonserver, release, component, versionname, forceFlag, WorkAreaName, remarks, commonreleases, filetype, parttype, retainlockFlag, configFields, effectiveUserID, VerboseFlag
PartCheckOut	0	partpathName, temporaryfileonserver, release, forceFlag, WorkAreaName, parttype, effectiveUserID, TeamcUserID, VerboseFlag
	1	partpathName, temporaryfileonserver, release, filetype, component, versionname, forceFlag, workareaname, parttype, configFields, effectiveUserID, VerboseFlag
	2	partpathName, temporaryfileonserver, release, filetype, component, versionname, forceFlag, workareaname, parttype, configFields, effectiveUserID, VerboseFlag
PartChildInfo	0	partpathName, release, versionname, WorkAreaName, displaytype, relationtoparent, parttype, effectiveUserID, TeamcUserID, VerboseFlag
	1	partpathName, release, versionname, WorkAreaName, displaytype, relationtoparent, parttype, effectiveUserID, VerboseFlag
	2	partpathName, release, versionname, WorkAreaName, displaytype, relationtoparent, parttype, effectiveUserID, VerboseFlag

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
PartConnect	0	partpathName, release, WorkAreaName, parentname, relationtoparent, parttype, parenttype, effectiveUserID, TeamcUserID, VerboseFlag
	1	partpathName, release, WorkAreaName, component, parentname, relationtoparent, parttype, parenttype, effectiveUserID, VerboseFlag
	2	partpathName, release, WorkAreaName, component, parentname, relationtoparent, parttype, parenttype, effectiveUserID, VerboseFlag
PartDelete	0	partpathName, release, forceFlag, WorkAreaName, commonFlag, parttype, effectiveUserID, TeamcUserID, VerboseFlag
	1	partpathName, release, forceFlag, WorkAreaName, commonFlag, component, parttype, effectiveUserID, VerboseFlag
	2	partpathName, release, forceFlag, WorkAreaName, commonFlag, component, parttype, effectiveUserID, VerboseFlag
PartDisconnect	0	partpathName, release, WorkAreaName, parentname, parttype, parenttype, effectiveUserID, TeamcUserID, VerboseFlag
	1	partpathName, release, WorkAreaName, component, parentname, parttype, parenttype, effectiveUserID, VerboseFlag
	2	partpathName, release, WorkAreaName, component, parentname, parttype, parenttype, effectiveUserID, VerboseFlag
PartExtract	0	partpathName, temporaryfileonserver, release, nokeysFlag, WorkAreaName, versionname, parttype, effectiveUserID, TeamcUserID, VerboseFlag
	1	partpathName, temporaryfileonserver, release, nokeysFlag, WorkAreaName, versionname, component, parttype, configFields, effectiveUserID, VerboseFlag
	2	partpathName, temporaryfileonserver, release, nokeysFlag, WorkAreaName, versionname, component, parttype, configFields, effectiveUserID, VerboseFlag
PartLink	0	partpathName, sourcerelease, release, sourceworkareaname, sourceversion, parttype, targetworkareaname, effectiveUserID, TeamcUserID, VerboseFlag
	1	partpathName, sourceworkareaname, sourcerelease, targetrelease, sourceversion, component, parttype, targetworkareaname, effectiveUserID, VerboseFlag
	2	partpathName, sourceworkareaname, sourcerelease, targetrelease, sourceversion, component, parttype, targetworkareaname, effectiveUserID, VerboseFlag
PartLock	0	partpathName, temporaryfileonserver, release, forceFlag, WorkAreaName, parttype, effectiveUserID, TeamcUserID, VerboseFlag
	1	partpathName, release, forceFlag, WorkAreaName, filetype, component, versionname, parttype, configFields, effectiveUserID, VerboseFlag
	2	partpathName, release, forceFlag, WorkAreaName, filetype, component, versionname, parttype, configFields, effectiveUserID, VerboseFlag
PartModify	0	partpathName, release, newcomponent, newfMode, configFields, WorkAreaName, filetype, parsername, buildername, buildparameters, parttype, temporaryfilename, effectiveUserID, TeamcUserID, VerboseFlag

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
	1	partpathName, release, oldcomponent, newcomponent, oldfMode, newfMode, configFields, WorkAreaName, dateoflastupdate, filetype, parsername, buildname, buildparameters, parttype, temporaryFlag, effectiveUserID, VerboseFlag
	2	partpathName, release, oldcomponent, newcomponent, oldfMode, newfMode, configFields, WorkAreaName, dateoflastupdate, filetype, parsername, buildname, buildparameters, parttype, temporaryFlag, effectiveUserID, VerboseFlag
	0	partpathName, release, forceFlag, WorkAreaName, commonFlag, parttype, effectiveUserID, TeamcUserID, VerboseFlag
PartRecreate	1	partpathName, release, forceFlag, WorkAreaName, commonFlag, component, olddropDate, parttype, effectiveUserID, VerboseFlag
	2	partpathName, release, forceFlag, WorkAreaName, commonFlag, component, olddropDate, parttype, effectiveUserID, VerboseFlag
	0	partpathName, sourcerelease, release, sourceworkareaname, sourceversion, parttype, targetworkareaname, effectiveUserID, TeamcUserID, VerboseFlag
PartRefresh	1	partpathName, sourceworkareaname, sourcerelease, targetrelease, sourceversion, component, parttype, targetworkareaname, effectiveUserID, VerboseFlag
	2	partpathName, sourceworkareaname, sourcerelease, targetrelease, sourceversion, component, parttype, targetworkareaname, effectiveUserID, VerboseFlag
	0	partpathName, release, nuPartPathName, forceFlag, WorkAreaName, commonFlag, parttype, effectiveUserID, TeamcUserID, VerboseFlag
PartRename	1	partpathName, release, nuPartPathName, forceFlag, WorkAreaName, commonFlag, component, parttype, effectiveUserID, VerboseFlag
	2	partpathName, release, nuPartPathName, forceFlag, WorkAreaName, commonFlag, component, parttype, effectiveUserID, VerboseFlag
	0	partpathName, release, forceFlag, WorkAreaName, commonFlag, parttype, creatChangeFlag, effectiveUserID, TeamcUserID, VerboseFlag
PartTouch	1	partpathName, release, forceFlag, WorkAreaName, commonFlag, component, parttype, effectiveUserID, VerboseFlag
	2	partpathName, release, forceFlag, WorkAreaName, commonFlag, component, parttype, effectiveUserID, VerboseFlag
	0	partpathName, release, forceFlag, WorkAreaName, commonFlag, parttype, effectiveUserID, TeamcUserID, VerboseFlag
PartUndo	1	partpathName, release, forceFlag, WorkAreaName, commonFlag, component, versionname, parttype, effectiveUserID, VerboseFlag
	2	partpathName, release, forceFlag, WorkAreaName, commonFlag, component, versionname, parttype, effectiveUserID, VerboseFlag
	0	partpathName, WorkAreaName, release, parttype, effectiveUserID, TeamcUserID, VerboseFlag
PartUnlock		

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
	1	partpathName, WorkAreaName, release, component, parttype, configFields, effectiveUserID, VerboseFlag
	2	partpathName, WorkAreaName, release, component, parttype, configFields, effectiveUserID, VerboseFlag
PartView	0	partpathName, release, versionname, WorkAreaName, displaytype, parttype, effectiveUserID, TeamcUserID, VerboseFlag
	1	partpathName, release, versionname, WorkAreaName, displaytype, parttype, effectiveUserID, VerboseFlag
	2	partpathName, release, versionname, WorkAreaName, displaytype, parttype, effectiveUserID, VerboseFlag
	0	partpathName, release, versionname, WorkAreaName, parttype, effectiveUserID, TeamcUserID, VerboseFlag
PartViewmsg	1	partpathName, release, component, versionname, WorkAreaName, parttype, effectiveUserID, VerboseFlag
	2	partpathName, release, component, versionname, WorkAreaName, parttype, effectiveUserID, VerboseFlag
PrereqCreate	0	release, primeworkareaname, secondworkareaname, effectiveUserID, TeamcUserID, VerboseFlag
	1	release, primeworkareaname, secondworkareaname, effectiveUserID, VerboseFlag
	2	release, primeworkareaname, secondworkareaname, effectiveUserID, VerboseFlag
	0	release, primeworkareaname, secondworkareaname, effectiveUserID, TeamcUserID, VerboseFlag
PrereqDelete	1	release, primeworkareaname, secondworkareaname, effectiveUserID, VerboseFlag
	2	release, primeworkareaname, secondworkareaname, effectiveUserID, VerboseFlag
	0	release, component, newreleaseprocess, environmentname, testersname, ApproverName, description, releaseowner, autoprune, developmentmode, releasedatabasename, outputversions, StandardFields, configFields, effectiveUserID, TeamcUserID, VerboseFlag
	1	release, component, newreleaseprocess, environmentname, testersname, ApproverName, description, releaseowner, effectiveUserID, VerboseFlag
ReleaseCreate	2	release, component, newreleaseprocess, environmentname, testersname, ApproverName, description, releaseowner, effectiveUserID, VerboseFlag
	0	release, effectiveUserID, TeamcUserID, VerboseFlag
	1	release, effectiveUserID, VerboseFlag
	2	release, effectiveUserID, VerboseFlag
ReleaseDelete	0	release, root, node, nokeysFlag, committedFlag, date, fmask, dmask, uid, gid, crlfFlag, versionname, effectiveUserID, TeamcUserID, VerboseFlag
	1	release, root, node, nokeysFlag, committedFlag, date, fmask, dmask, uid, gid, crlfFlag, versionname, effectiveUserID, TeamcUserID, VerboseFlag
ReleaseExtract	0	release, root, node, nokeysFlag, committedFlag, date, fmask, dmask, uid, gid, crlfFlag, versionname, effectiveUserID, TeamcUserID, VerboseFlag
	1	release, root, node, nokeysFlag, committedFlag, date, fmask, dmask, uid, gid, crlfFlag, versionname, effectiveUserID, TeamcUserID, VerboseFlag

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
	1	release, root, node, nokeysFlag, committedFlag, date, fmask, dmask, uid, gid, crlfFlag, versionname, effectiveUserID, VerboseFlag
	2	release, root, node, nokeysFlag, committedFlag, date, fmask, dmask, uid, gid, crlfFlag, versionname, effectiveUserID, VerboseFlag
	0	release, FromRelease, WorkAreaName, newworkareaname, fromversionname, effectiveUserID, TeamcUserID, VerboseFlag
ReleaseLink	1	release, FromRelease, WorkAreaName, fromworkareaname, fromversionname, effectiveUserID, VerboseFlag
	2	release, FromRelease, WorkAreaName, fromworkareaname, fromversionname, effectiveUserID, VerboseFlag
	0	release, newrelease, component, description, newreleaseprocess, environmentname, testersname, ApproverName, NewOwner, autoprun, outputversions, StandardFields, configFields, effectiveUserID, TeamcUserID, VerboseFlag
ReleaseModify	1	release, newrelease, oldcomponent, newcomponent, olddescription, newdescription, oldreleaseprocess, newreleaseprocess, environmentname, testersname, ApproverName, OldOwnerName, NewOwner, dateoflastupdate, effectiveUserID, VerboseFlag
	2	release, newrelease, component, description, newreleaseprocess, environmentname, testersname, ApproverName, NewOwner, effectiveUserID, VerboseFlag
	0	release, versionname, effectiveUserID, TeamcUserID, VerboseFlag
ReleasePrune	1	release, versionname, effectiveUserID, VerboseFlag
	2	release, versionname, effectiveUserID, VerboseFlag
	0	release, environmentname, testersname, ApproverName, effectiveUserID, TeamcUserID, VerboseFlag
ReleaseRecreate	1	release, lastdropdate, environment, testersname, ApproverName, effectiveUserID, VerboseFlag
	2	release, lastdropdate, environment, testersname, ApproverName, effectiveUserID, VerboseFlag
	0	release, reporttype, effectiveUserID, TeamcUserID, VerboseFlag
ReleaseView	1	release, reporttype, effectiveUserID, VerboseFlag
	2	release, reporttype, effectiveUserID, VerboseFlag
	0	viewname, reportcriteria, parent, release, WorkAreaName, versionname, reporttype, parenttype, effectiveUserID, TeamcUserID, VerboseFlag
Report	1	viewname, reportcriteria, parent, effectiveUserID, VerboseFlag
	2	viewname, reportcriteria, parent, effectiveUserID, VerboseFlag



<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
SizeAccept	0	WorkAreaName, component, release, sizetext, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, component, release, sizetext, sizetype, effectiveUserID, VerboseFlag
	2	WorkAreaName, component, release, sizetext, sizetype, effectiveUserID, VerboseFlag
SizeAssign	0	WorkAreaName, component, release, NewOwner, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, component, release, NewOwner, sizetype, effectiveUserID, VerboseFlag
	2	WorkAreaName, component, release, NewOwner, sizetype, effectiveUserID, VerboseFlag
SizeCreate	0	WorkAreaName, component, release, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, component, release, sizetype, effectiveUserID, VerboseFlag
	2	WorkAreaName, component, release, sizetype, effectiveUserID, VerboseFlag
SizeDelete	0	WorkAreaName, component, release, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, component, release, sizetype, effectiveUserID, VerboseFlag
	2	WorkAreaName, component, release, sizetype, effectiveUserID, VerboseFlag
SizeReject	0	WorkAreaName, component, release, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, component, release, sizetype, effectiveUserID, VerboseFlag
	2	WorkAreaName, component, release, sizetype, effectiveUserID, VerboseFlag
TestAbstain	0	WorkAreaName, TesterName, release, environmentname, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, DefectOrFeatureName, TesterName, release, environmentname, type, effectiveUserID, VerboseFlag
	2	WorkAreaName, DefectOrFeatureName, TesterName, release, environmentname, type, effectiveUserID, VerboseFlag
TestAccept	0	WorkAreaName, TesterName, release, environmentname, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, DefectOrFeatureName, TesterName, release, environmentname, type, effectiveUserID, VerboseFlag
	2	WorkAreaName, DefectOrFeatureName, TesterName, release, environmentname, type, effectiveUserID, VerboseFlag
TestAssign	0	WorkAreaName, OldOwner, NewOwner, release, environmentname, effectiveUserID, TeamcUserID, VerboseFlag

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
	1	WorkAreaName, OldOwner, NewOwner, release, environmentname, workareaType, effectiveUserID, VerboseFlag
	2	WorkAreaName, OldOwner, NewOwner, release, environmentname, workareaType, effectiveUserID, VerboseFlag
	0	WorkAreaName, TesterName, release, environmentname, effectiveUserID, TeamcUserID, VerboseFlag
TestReject	1	WorkAreaName, DefectOrFeatureName, TesterName, release, environmentname, type, effectiveUserID, VerboseFlag
	2	WorkAreaName, DefectOrFeatureName, TesterName, release, environmentname, type, effectiveUserID, VerboseFlag
	0	login, usersfullname, area, sendmailaddress, superuserprivilegeFlag, configFields, effectiveUserID, TeamcUserID, VerboseFlag
UserCreate	1	login, usersfullname, area, sendmailaddress, superuserprivilegeFlag, configFields, effectiveUserID, VerboseFlag
	2	login, usersfullname, area, sendmailaddress, superuserprivilegeFlag, configFields, effectiveUserID, VerboseFlag
	0	login, effectiveUserID, TeamcUserID, VerboseFlag
UserDelete	1	login, usersfullname, effectiveUserID, VerboseFlag
	2	login, usersfullname, effectiveUserID, VerboseFlag
	0	login, newlogin, newusersfullname, newarea, newuserssendmailaddress, newsuperuserprivilegeFlag, configFields, passwordlength, oldpassword, newpassword, effectiveUserID, TeamcUserID, VerboseFlag
UserModify	1	login, newlogin, oldusersfullname, newusersfullname, oldarea, newarea, oldsendmailaddress, newsendmailaddress, oldsuperuserprivilegeFlag, newsuperuserprivilegeFlag, configFields, dateoflastupdate, effectiveUserID, VerboseFlag
	2	login, newlogin, newusersfullname, newarea, newuserssendmailaddress, newsuperuserprivilegeFlag, configFields, effectiveUserID, VerboseFlag
	0	login, effectiveUserID, TeamcUserID, VerboseFlag
UserRecreate	1	login, usersfullname, olddropDate, effectiveUserID, VerboseFlag
	2	login, usersfullname, olddropDate, effectiveUserID, VerboseFlag
	0	login, displaytype, effectiveUserID, TeamcUserID, VerboseFlag
UserView	1	login, displaytype, effectiveUserID, VerboseFlag
	2	login, displaytype, effectiveUserID, VerboseFlag
	0	WorkAreaName, TesterName, effectiveUserID, TeamcUserID, VerboseFlag
VerifyAbstain	1	WorkAreaName, TesterName, type, effectiveUserID, VerboseFlag

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
VerifyAccept	2	WorkAreaName, TesterName, type, effectiveUserID, VerboseFlag
	0	WorkAreaName, TesterName, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, TesterName, type, effectiveUserID, VerboseFlag
VerifyAssign	2	WorkAreaName, TesterName, type, effectiveUserID, VerboseFlag
	0	WorkAreaName, OldOwner, NewOwner, effectiveUserID, TesterName, VerboseFlag
	1	WorkAreaName, OldOwner, NewOwner, type, effectiveUserID, VerboseFlag
VerifyReject	2	WorkAreaName, OldOwner, NewOwner, type, effectiveUserID, VerboseFlag
	0	WorkAreaName, TesterName, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, TesterName, type, effectiveUserID, VerboseFlag
WorkAreaAssign	2	WorkAreaName, TesterName, type, effectiveUserID, VerboseFlag
	0	release, WorkAreaName, NewOwner, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, release, DefectOrFeatureName, NewOwner, workareaType, effectiveUserID, VerboseFlag
WorkAreaCancel	2	WorkAreaName, release, DefectOrFeatureName, NewOwner, workareaType, effectiveUserID, VerboseFlag
	0	release, WorkAreaName, forceFlag, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, release, DefectOrFeatureName, workareaType, effectiveUserID, VerboseFlag
WorkAreaCheck	2	WorkAreaName, release, DefectOrFeatureName, workareaType, effectiveUserID, VerboseFlag
	0	release, WorkAreaName, driver, effectiveUserID, TeamcUserID, VerboseFlag
	1	release, WorkAreaName, drivername, workareaType, effectiveUserID, VerboseFlag
WorkAreaCommit	2	release, WorkAreaName, drivername, workareaType, effectiveUserID, VerboseFlag
	0	release, WorkAreaName, forceFlag, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, release, DefectOrFeatureName, workareaType, effectiveUserID, VerboseFlag
WorkAreaComple	2	WorkAreaName, release, DefectOrFeatureName, workareaType, effectiveUserID, VerboseFlag
	0	release, WorkAreaName, forceFlag, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, release, DefectOrFeatureName, workareaType, effectiveUserID, VerboseFlag
	2	WorkAreaName, release, DefectOrFeatureName, workareaType, effectiveUserID, VerboseFlag

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
WorkAreaCreate	0	release, WorkAreaName, DefectOrFeatureName, target, workareaOwner, StandardFields, configFields, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, release, DefectOrFeatureName, target, workareaOwner, workareaType, effectiveUserID, VerboseFlag
	2	WorkAreaName, release, DefectOrFeatureName, target, workareaOwner, workareaType, effectiveUserID, VerboseFlag
WorkAreaFix	0	release, WorkAreaName, forceFlag, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, release, DefectOrFeatureName, workareaType, effectiveUserID, VerboseFlag
	2	WorkAreaName, release, DefectOrFeatureName, workareaType, effectiveUserID, VerboseFlag
WorkAreaFreeze	0	release, WorkAreaName, effectiveUserID, TeamcUserID, VerboseFlag
	1	release, WorkAreaName, workareatarget, workareaType, effectiveUserID, VerboseFlag
	2	release, WorkAreaName, workareatarget, workareaType, effectiveUserID, VerboseFlag
WorkAreaIntegra	0	release, WorkAreaName, forceFlag, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, release, DefectOrFeatureName, workareaType, effectiveUserID, VerboseFlag
	2	WorkAreaName, release, DefectOrFeatureName, workareaType, effectiveUserID, VerboseFlag
WorkAreaModify	0	release, WorkAreaName, newtarget, StandardFields, configFields, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, release, oldtarget, newtarget, workareaType, effectiveUserID, VerboseFlag
	2	WorkAreaName, release, oldtarget, newtarget, workareaType, effectiveUserID, VerboseFlag
WorkAreaRefresh	0	release, WorkAreaName, source, effectiveUserID, TeamcUserID, VerboseFlag
	1	release, WorkAreaName, source, workareaType, effectiveUserID, VerboseFlag
	2	release, WorkAreaName, source, workareaType, effectiveUserID, VerboseFlag
WorkAreaTest	0	release, WorkAreaName, forceFlag, effectiveUserID, TeamcUserID, VerboseFlag
	1	WorkAreaName, release, DefectOrFeatureName, workareaType, effectiveUserID, VerboseFlag
	2	WorkAreaName, release, DefectOrFeatureName, workareaType, effectiveUserID, VerboseFlag
WorkAreaUndo	0	release, WorkAreaName, effectiveUserID, TeamcUserID, VerboseFlag

<b>TeamConnection action</b>	<b>Exit ID</b>	<b>Parameters passed to the user exit program (see page 413 for definitions)</b>
	1	release, WorkAreaName, target, workareaType, effectiveUserID, VerboseFlag
	2	release, WorkAreaName, target, workareaType, effectiveUserID, VerboseFlag
WorkAreaView	0	release, WorkAreaName, reporttype, effectiveUserID, TeamcUserID, VerboseFlag
	1	release, WorkAreaName, reporttype, workareaType, effectiveUserID, VerboseFlag
	2	release, WorkAreaName, reporttype, workareaType, effectiveUserID, VerboseFlag

## User exit parameter definitions

The following list provides definitions for most of the parameters passed to user exit programs. Parameters are listed in alphabetical order. Parameter names are in lowercase, except where they are the name of a field in a TeamConnection database table. For more information on these and other parameters, refer to the *Commands Reference*.

### **abstract**

Defect or feature abstract.

### **alternateversion**

Specifies the name of a version of a driver, release, or work area where the conflicting version of a part is visible.

### **answer**

Specifies the reason for an action taken on a defect.

### **ApproverName**

Approver's TeamConnection user ID.

**area** Department or area in which the user works.

### **authority**

Specifies a user's authority group.

### **autoprune**

Whether or not to automatically prune work areas that have not been integrated with the release. Valid values are yes and no.

### **buildername**

The name of the builder used to create an output part.

### **buildmode**

The mode in which the build runs. The following values are valid:

- 1** force
- 2** normal
- 3** unconditional
- 4** report

### **buildparameters**

Specifies the parameters passed to the build script.

### **cancelFlag**

This flag is used with PartBuild actions to cancel a build request.

**clienthostname**

The name of the system where the client command originated.

**clientportname**

The port number used for the build pool.

**committedFlag**

This flag is used with ReleaseExtract and ReleaseLink actions to specify whether the user wants the last committed (as opposed to the current) versions of parts in the release. A value of 0 means to use the current version; 1 means to use the last committed version.

**commonFlag**

Indicates whether the part is common with other releases or not. A value of 0 indicates no, 1 indicates yes.

**commonreleases**

For a common part, this parameter specifies the other releases the part is common with (on the partCheckIn action). Release names are separated by blanks.

**component**

Specifies the name of a component.

**componentprocess**

Specifies the process to be used for a component in CompCreate actions.

**condition**

This parameter is used with the value parameter to determine if a build event was successful.

**configFields**

This parameter has the format:

```
attribute name    content
attribute name    content
:
null string
```

For exit ID 0, the attribute name can appear in abbreviated form, as it is not processed by TeamConnection.

**creatChangeFlag**

This flag is used by PartTouch to specify the write permissions of the part: 0200 permits write by the owner; 0000 does not allow write. This parameter is defined numerically, in octal notation. The fMode code is constructed by combing the logical OR of the following values:

```
4000    setuid
2000    setgid
0400    Permits read by owner
0200    Permits write by owner
0100    Permits execute or search by owner
0040    Permits read by group
0020    Permits write by group
0010    Permits execute or search by group
0004    Permits read by all others
```

**0002** Permits write by all others

**0001** Permits execute or search by all others

For example, 0755 would permit read, write, and execute for the owner and read and execute for all others. .

**crLfFlag**

This flag is used by DriverExtract and ReleaseExtract to handle crlf conversions when extracting Intel-based files to a UNIX-based platform.

**date** Enables you to extract only files from a release that are older than the specified date.

**dateoflastupdate**

Specifies the date in modify actions.

**defectname**

Indicates the name of the defect.

**DefectOrFeatureName**

Indicates the name of the defect or feature for approval record, fix record, test record, driver member, or work area actions.

**description**

Specifies a description of an object.

**detailfilename**

Specifies the file in which all build messages for a part are collected.

**developmentmode**

Valid values are serial and concurrent.

**displaytype**

This parameter is used on all view actions. The type of view format requested, where:

- |          |         |
|----------|---------|
| <b>0</b> | stanza  |
| <b>1</b> | raw     |
| <b>2</b> | table   |
| <b>3</b> | long    |
| <b>4</b> | process |

**dmask**

Specifies the read, write, and execute directory permissions for the extracted parts in octal notation.

**driver, drivename**

Specifies the name of the driver for defect, driver, driver member, and work area actions.

**driverstate**

Values can be working, integrate, commit, or complete.

**drivertype**

Specified by the user when a driver is created, for example, development, production, or prototype.

**effectiveUserID**

The TeamConnection user ID that initiated the transaction. This is the value of the TC\_BECOME environment variable or the -become attribute flag. In

OS/2, Windows 3.1, and Windows 95 environments, if this variable is not set and the -become attribute is not specified, it is the value of the TC\_USER environment variable.

**environment, environmentname**

Specifies the environment in which the testing is to be done if the test subprocess is included in the release process. (The tester/environment name combination becomes an entry on the environment list for the release.)

**ExtractType**

Indicates whether this is a full or delta driver extract. 0 indicates delta; 1 indicates full.

**featurename**

Specifies the name of the feature for feature actions.

**filenameonclient**

The name of the source file from which a TeamConnection part is created using the PartAdd or PartCheckin action.

**filetype**

Specifies one of the following file types with the PartAdd action:

- 0** none
- 1** text
- 2** binary

Part type for the other part actions is text for text parts, and binary for binary parts.

**fmask** Specifies the read, write, and execute file permissions for the extracted parts in octal notation. Refer to the *Commands Reference* for details.

**fMode** Specifies the write permissions of the part: 0200 permits write by the owner; 0000 does not allow write. This parameter is defined numerically, in octal notation. The fMode code is constructed by combing the logical OR of the following values:

- 4000** setuid
- 2000** setgid
- 0400** Permits read by owner
- 0200** Permits write by owner
- 0100** Permits execute or search by owner
- 0040** Permits read by group
- 0020** Permits write by group
- 0010** Permits execute or search by group
- 0004** Permits read by all others
- 0002** Permits write by all others
- 0001** Permits execute or search by all others

For example, 0755 would permit read, write, and execute for the owner and read and execute for all others.



**forceFlag**

Indicates whether the force option was chosen on part actions (0 indicates no and 1 indicates yes). The force option is used to force a break between common parts when using PartLock, PartCheckOut, PartCheckIn, PartDelete, PartRecreate, PartRename, and PartUndo actions.

**FromRelease**

Specifies the name of the release to be linked from in ReleaseLink actions.

**fromversionname**

Specifies the version of the release to be linked from in ReleaseLink actions.

**fromworkareaname**

Specifies the name of the work area to be linked from in ReleaseLink actions.

**gid**

Specifies ownership of extracted parts by identifying the internal number that uniquely identifies the group to the system.

**interestgroupname**

Specifies the name of an interest group in a NotifyCreate or NotifyDelete action.

**lastdropdate**

Specifies the date on which a release to be recreated using the ReleaseRecreate action was deleted.

**login**

The system login ID for a user. In single-user environments, such as OS/2, Windows 3.1 and Windows 95, this parameter is the TC\_USER environment variable.

**login@hostname**

Specifies a host list entry in HostCreate and HostDelete actions.

**longFlag**

Indicates whether the -long flag is specified or not specified. A value of 0 indicates not specified; 1 indicates specified. The -long flag is available on some of the view actions and is used to display more detailed information of the object being viewed. The -long flag on the driverCheck action displays details about prerequisites and corequisites.

**name**

Specifies the name of a builder in Builder actions. In the CompModify action, this parameter specifies the current component name.

**newarea**

Specifies a new area or department in which a user works.

**newcomponent**

Specifies a new name for a component.

**newcomponentprocess**

Specifies a new process to be used for a component.

**newdefectname**

Specifies a new name for a defect in a DefectModify action.

**newdescription**

Specifies a new description for an object.

**newdrivername**

Specifies a new name for a driver in a DriverModify action.

**newdrivertype**

Specifies a new type for the driver in DriverModify actions. Valid types include development, production, or prototype.

**newfeaturename**

Specifies a new name for a feature in a FeatureModify action.

**newfMode**

Specifies a new write permission for a PartModify action. See **fmode** for a list of values.

**newlogin**

Specifies a new login ID for a user.

**NewOwner**

Specifies the new owner of an object in actions that create, modify, or assign owners to objects.

**newrelease**

Specifies a new release name for ReleaseModify actions.

**newreleaseprocess**

Specifies the release process to be used for ReleaseCreate or ReleaseModify actions.

**newsendmailaddress**

Specifies a new email address for a user's notification messages.

**newsuperuserprivilegeflag**

Specifies the user's superuser status for UserModify actions. Specify 0 to deny superuser status or 1 to grant superuser status.

**newtarget**

Specifies a new target for work areas.

**newtestersname**

Specifies the full name of a new tester in an EnvModify action.

**newtype**

Specifies a new driver type for DriverModify actions. Valid types include development, production, or prototype.

**newusersfullname**

Specifies the new full name for a user in UserModify actions.

**newuserssendmailaddress**

Specifies the new mail address for a user in UserModify actions.

**newworkareaname**

Specifies the work area to be linked to in ReleaseLink actions.

**node** Specifies a remote host on which to place the extracted part tree.

**nokeysFlag**

For extract actions, indicates whether you want to substitute assigned values in place of keywords imbedded in the extracted parts. 0 means not to substitute assigned values; 1 means to substitute assigned values.

**numberofworkareas**

Specifies the number of work areas to be deleted in a MemberDelete action.

**nuPartPathName**

Specifies the new path name for PartRename actions.

**oldcomponentprocess**

Specifies the process of a component to be modified.

**olddescription**

Specifies the description of an object to be modified.

**olddropDate**

Specifies the date on which a component, part, or user to be recreated using the CompRecreate, PartRecreate, or UserRecreate action was deleted.

**oldfMode**

Specifies the old write permission for a PartModify action. See **fmode** for a list of values.

**OldOwner**

Specifies the old owner of an object in actions that modify or assign owners to objects.

**oldreleaseprocess**

Specifies the old release process to be changed by a ReleaseModify action.

**oldsendmailaddress**

Specifies an email address to be changed for a user's notification messages.

**oldsuperuserprivilegeflag**

Specifies the user's superuser status to be changed by a UserModify action. Specify 0 if the user currently does not have superuser status or 1 if he or she currently does have superuser status.

**oldtarget**

Specifies a target to be changed for work areas.

**oldtype**

Specifies the driver type to be changed by a DriverModify action. Valid types include development, production, or prototype.

**oldusersfullname**

Specifies the full name for the user to be changed by a UserModify action.

**originaldefectname**

The name of a defect for which the current defect is a duplicate.

**originalfeaturename**

The name of a feature for which the current feature is a duplicate.

**originator**

The TeamConnection user ID of the user who opens the defect or feature.

**outputversions**

Specifies the number of versions of build output parts to be retained in ReleaseCreate or ReleaseModify actions.

**owner** Specifies the component owner in CompCreate actions.

**parent**

Specifies the parent of the part to generate a report on using the ReportView —partview action.

**parentcomponent**

Specifies the parent component for Component actions.

**parentname**

Specifies the parent of a part in a build tree in PartAdd, PartConnect, and PartDisconnect actions.

**parenttype**

Specifies the part type of the parent of a part in a build tree in PartAdd, PartConnect, and PartDisconnect actions. In Report actions, this parameter specifies the part type of the parent of the part to generate a report on using the ReportView —partview action.

**parsername**

Specifies the name of the parser used to create an output part.

**parsercommand**

Specifies the command file you want to associate with the parser. This can be an .exe, a .com, a .cmd, or a .bat file. The executable file needs to be in the execution path of the TeamConnection family server.

**partpathName**

Specifies the path name of a part in Part actions.

**parttype**

Specifies the type of a part, such as TcPart or vgdata.

**pathName**

Specifies the path name of parts in Collision actions.

**paths** Specifies a concatenated set of paths that define where the parser looks for parts when processing the set of dependencies returned from the command file. These dependencies come in two types:

- A dependency in which the file is stored in the TeamConnection database. For example, hello.c includes hello.h, and both files are stored in the TeamConnection database. During a build, these dependencies must be extracted to a path accessible by the build processor.
- A dependency on a file that is not stored in the TeamConnection database. An example of such a dependency is stdio.h, which is typically stored in a compiler's include path and not in the TeamConnection database.

**poolname**

Specifies the build pool used to build a part.

**prefix** Defect or feature prefix.

**primeworkareaname**

Prime corequisite work area name.

**process**

Specifies the component process in CompModify actions.

**processoroptions**

Parameters specified for passing to a builder upon builder -create.

**reference**

Defect or feature reference.

**relationtoparent**

How a part is related to its parent in the build tree, where:

- 1 input
- 2 output
- 3 dependent

**release**

Name of the release.

**releasedatabasename**

Name of a separate database for the part data (the contents of each part) in a release.

**releaseowner**

Specifies the owner of a release.

**remarks**

For defect or feature actions, this is defect remarks or feature remarks. For part actions, this is part remarks added when a new version is created.

**reportcriteria**

The criteria entered as the -where clause for a Report action.

**reporttype**

The type of report format requested, where:

- 0 stanza
- 1 raw
- 2 table
- 3 long
- 4 process

User exit messages are not displayed if the -raw format is selected.

**retainlockFlag**

Specifies that a part is to remain locked after it is checked in.

**root**

This is the specified directory on the designated host where the extracted part tree is to be placed.

**script**

Specifies the name of the build script.

**secondworkareaname**

Second corequisite work area name.

**sendmailaddress**

The e-mail address to which a user's notification messages are sent.

**severity**

Defect severity driver.

**sizetext**

The sizing information for a defect or feature.

**sizetype**

Specifies whether the sizing record is associated with a defect or a feature.

**source**

Specifies the name of a work area with which another work area is refreshed.

**sourcerelease**

Specifies the original release of a part to be linked using the PartLink action.

**sourceversion**

Specifies the original version of a part to be linked using the PartLink action.

**sourceworkareaname**

Specifies the original work area of a part to be linked using the PartLink action.

**StandardFields**

Contains any fields abbreviated by users, requiring interpretation and verification on the TeamConnection server. Actions with configurable fields allow for the ambiguity in parameter names that requires intervention by the server.

**state** Specifies the state of parts in Collision actions.

**superuserprivilegeflag**

A value of yes indicates on; a value of no indicates off.

**target** Specifies the value of the work area target field for a WorkareaUndo action.

**targetenvironment**

Specifies the environment for which build output is generated.

**targetrelease**

Specifies the new release of a part to be linked using the PartLink action.

**targetworkareaname**

Specifies the new work area of a part to be linked using the PartLink action.

**TeamcUserID**

The user's TeamConnection user ID on the client workstation. In AIX, HP-UX, and Windows NT environments, this is the login ID. In OS/2, Windows 3.1, and Windows 95 environments, this is the value of the TC\_USER environment variable.

**temporaryfilename**

Indicates if the -temporary flag is used on a Part -modify command.

**temporaryfileonserver**

For some part actions, the contents of the file on the client are copied to a temporary file on the server. This parameter is the name for the temporary file on the server.

**temporaryFlag**

Indicates the part is a temporary part on PartAdd and PartModify actions.

**testersname, TesterName**

Specifies the full name of the person responsible for testing an object.

**timeout**

Specifies the amount of time that the build processor waits for a build script to complete before assuming a failure has occurred. The default is 1440 minutes (24 hours).

**transmitFlag**

Indicates whether the builder part is to be copied from the client to the server or not. Specify 0 or 1.

**type** Specifies whether the sizing record is associated with a defect or a feature on Test actions.

**typename**

Specifies the type of parts being handled by Collision actions.

**uid**

Specifies ownership of extracted parts by identifying the internal number that uniquely identifies the user to the system.

**usersfullName**

Specifies the full name of a user.

**value** This parameter is used with the condition parameter to determine if a build event was successful.

**VerboseFlag**

Specifies that you want to see a confirmation message after you issue this action. 0 indicates off; 1 indicates on. The user exit program can use this flag to include confirmation or status messages only when the -verbose flag is on.

**versionname**

Part version name.

**viewname**

The name of the view (for example, partView) that is being reported on.

**WorkAreaName**

Specifies the name of a work area.

**workareaOwner**

Specifies the owner of a work area.

**workareastate**

The value can be approve, fix, integrate, commit, test, or complete.

**workareatarget**

Specifies the target field used when creating or modifying a workarea in WorkAreaFreeze actions.

**workareaType**

The value can be defect or feature.





## Appendix D. Environment Variables

You can set environment variables to describe the TeamConnection environment in which you are working. You are not required to set your TC\_FAMILY environment variable for the TeamConnection client command line interface. However, if the TC\_FAMILY environment variable is not set, the -family must be specified for every client command. See “Setting environment variables” on page 432 for more information about setting environment variables.

The names of the TeamConnection environment variables, the purpose they serve, the equivalent TeamConnection flag, the equivalent Settings notebook field, and the TeamConnection component that uses the environment variable are listed in the following table.

You can override the value you set for an environment variable by using the corresponding flag in a TeamConnection command. When an environment variable has a Settings notebook equivalent, TeamConnection uses the two as follows:

- The environment variable controls the command line interface.
- The Settings notebook controls the graphical user interface.

If there is no Settings notebook equivalent for the environment variable, then the environment variable takes effect regardless of the interface you are using.

To see a list of current environment variable settings, you can issue the following command from a command prompt:

```
teamc report -testServer
```

This command returns information like the following:

```
Connect to Family Name:      ptest
Server TCP/IP Name:         amachine.company.com
Server IP Address:          9.1.23.45
Server TCP/IP Port Number:  9999
```

```
Server Specific Information -----
Product Version:            3.0.0
Operating System:           AIX
Message catalog language:   English
Server Mode:                non-maintenance
Authentication Level:       HOST_ONLY
TC_RELEASE|v300
TC_FAMILY|ptest@amachine.company.com@9999
```

Table 25. TeamConnection environment variables

Environment variable	Purpose	Flag	Setting	Used by
LANG	Specifies the language-specific message catalog.			Client, family server
NLSPATH	Specifies the search path for locating message files.		NLS path	Client, family server

Table 25. TeamConnection environment variables (continued)

Environment variable	Purpose	Flag	Setting	Used by
OS_AS_SIZE	Specifies the size, in bytes, of the persistent storage region in the address space for each client. The default value is 128000000.			ObjectStore
OS_CACHE_SIZE	Specifies the size, in bytes, for the cache manager. The default value is 8192000. If you need to adjust this value, keep it to a value that can stay in physical memory rather than be swapped to disk.			ObjectStore
OS_NETWORK	Specifies the networking protocols used by ObjectStore servers and clients. Set during installation.			ObjectStore
OS_ROOTDIR	Specifies the path where the TeamConnection server is installed. Set during installation.			ObjectStore
OS_TMPDIR	Specifies where ObjectStore is to place temporary files. Set during installation. This variable is used only in Windows 32s environments.			ObjectStore
PATH	Specifies where tcadmin is to search for the family create utilities.			Family server
TC_ADMIN_DIR	Specifies the location of the default family options files, such as the authority groups file, security options file, and user exit files.			Administrator GUI
TC_ALLOWTRACKFIX	Allows users to add works areas in fix state to drivers.			Family server
TC_BECOME	Identifies the user ID you want to issue TeamConnection commands from, if the user ID differs from your login. You assume the access authority of the user ID you specify.	-become	Become user	Client, build server

Table 25. TeamConnection environment variables (continued)

Environment variable	Purpose	Flag	Setting	Used by
TC_BATCH_TXNS	Specifies the number of parts to be copied or extracted in each transaction for the following commands:workarea -integrate, driver -commit, workarea -refresh, driver -refresh, driverMember -add/-delete, release -link, release -extract, driver -extract The default value is ten, and most of the time this values does not need to be changed. Setting the value higher can improve performance in large databases. Setting the value lower can prevent the server running out of memory.			Family server
TC_BUILDENVIRONMENT	Specifies the build environment name, such as OS/2 or MVS. The value you specify here can be anything you like, but it must exactly match the environment specified for a builder in order for the builder to use this build agent. This value is case-sensitive.	-e		Client
TC_BUILDOPTS	Specifies build options for sending build log file messages to the screen, and setting the logging level. Possible values are TOSCREEN, and VERBOSE. If you do not specify any of these options, then the build server writes build messages to the build log file (teamcbld.log), and writes a minimum level of messages to the log file.	-c, -s, -l		Build server
TC_BUILDPOOL	Specifies the build pool name.	-pool	Pool	Build server
TC_BUILD_RSSBUILDS_FILE	Specifies the name of startup files to be used to provide information about build servers to the build process.			Build server
TC_BULKCHUNKSIZE	Identifies the size, in bytes, of the storage blocks used to store binary parts in ObjectStore. Do not use, unless you are trying to tune database performance with guidance from IBM service.			Family server

Table 25. TeamConnection environment variables (continued)

Environment variable	Purpose	Flag	Setting	Used by
TC_CASESENSE	Changes the case of the arguments in commands, not in queries.		Case	Client
TC_CATALOG	Specifies a specific file for the TeamConnection message catalog. Sometimes, depending upon the operating system environment, the catalog open command will only look in a particular directory for the catalog. If the host is running multiple versions of TeamConnection, this variable may be used. To set this environment variable, specify the file path name of the message catalog as in the following example: TC_CATALOG="//family/msgcat/teamc.cat"			Family server
TC_COMPONENT	Specifies the default component.	-component	Component	Client, make import tool
TC_DBPATH	Specifies the database directory path.			Family server
TC_DEADLOCKBEEP	Beeps if a deadlock occurs.			Family server
TC_DEADLOCK_DEBUG	Prints extra information to the build agent or family daemon if a deadlock occurs.			Family server
TC_DECACHE_COUNT	Decaches the ObjectStore cache manager after TeamConnection has processed the number of transactions specified. The default value is 256. Setting a lower value increases CPU usage. Setting a higher value increases memory and swapper usage.			Family server

Table 25. TeamConnection environment variables (continued)

Environment variable	Purpose	Flag	Setting	Used by
TC_DECACHE_WINDOW	Specifies how close to the TC_DECACHE_COUNT a decache will take place. The default value is 6. Many TeamConnection commands use many ObjectStore transactions to complete. Decaching after a TeamConnection command completes results in better performance. If you find that TeamConnection decaches during commands, you can adjust this value.			Family server
TC_FAMILY	Identifies the TeamConnection family you work with.	-family	Family	Build server, client, family server, make import tool
TC_FAMILY_SERVER_NOISY	Prints deadlock and recycling information to the family daemon. Use to debug deadlocks.			Family server
TC_MAKEIMPORTRULES	Specifies the name of the rules file that TeamConnection uses when importing the makefile data into TeamConnection. If you set this environment variable, then you do not have to use the /u option with the fhomigmk command. Specify the full path name of the rules file. If neither this environment variable nor the /u option is used, TeamConnection uses default rules.			Make import tool
TC_MAKEIMPORTTOP	Strips off the leading part of the directory name when importing parts into TeamConnection. For example, you have parts with the following directory structure: g:\octo\src\inc\. To create these parts without the g:\octo structure, you can set TC_MAKEIMPORTTOP=g:\octo before you invoke the make import tool. The parts created in TeamConnection have the directory structure of src\inc\.			Make import tool

Table 25. TeamConnection environment variables (continued)

Environment variable	Purpose	Flag	Setting	Used by
TC_MAKEIMPORTVERBOSE	Causes the -verbose flag to be added to part commands created by fhomigmk.			Make import tool
TC_MIGRATERULES	Specifies the name of a file containing the rules to be applied for migration of makefiles if the name is not supplied on the fhomigmk command line as a parameter.			Client
TC_NOTIFY_DAEMON	An alternate way of starting notifyd with the teamcd command. If you set this environment variable, then you do not have to use the -n option with the teamcd command. Specify the full path name of the mail exit to use with notifyd.			Family server
TC_RECYCLE_DEADLOCK_COUNT	Recycles the build agent or family daemon after it has received the number of commands specified. Set to any number greater than one. The default value is 100.			Family server
TC_RECYCLE_SERVICE_COUNT	Recycles the build agent or family daemon after it has executed the number of commands specified. Set to any number greater than ten. The default value is 100.			Family server
TC_RELEASE	Specifies a release.	-release	Release	Client, make import tool
TC_STACK_TRACE	Writes a stack trace in the syslog file if any messages are written to the log. The stack trace provides information that can help in debugging, including the DLL name, segment, and offset. This information can be matched with a MAP to determine which routine is causing a problem.			Family server
TC_SYSTEM_LOG	Specifies where syslog messages are to be written. Specify the full path name of the file to use for syslog messages. The default is syslog.log.			Family server
TC_TOP	Specifies the source directory.	-top	Top	Client

Table 25. TeamConnection environment variables (continued)

Environment variable	Purpose	Flag	Setting	Used by
TC_TRACE	Specifies the variable that lets the user designate which parts should be traced. You should modify this only when directed to do so by an IBM service person. Otherwise it is set to null. To trace all parts, specify TC_TRACE=*.			Client, family server, build server
TC_TRACEATTEMPTS	Specifies maximum number of failed trace attempts accepted before giving up. You should modify this only when directed to do so by an IBM service person.			Client, family server
TC_TRACEDELAY	Specifies the amount of time, in seconds, that TeamConnection waits, when a trace attempt fails, before attempting another trace. The default is 1 second. You should modify this only when directed to do so by an IBM service person.			Client, family server
TC_TRACEFILE	Specifies the output (part path and name) of the trace that the user designates using TC_TRACE. The default trace file name is tctrace. For the MVS build server, the default trace file is stdout.			Client, family server
TC_TRACESIZE	Specifies the maximum size of the trace file in bytes. If the maximum is reached, wrapping occurs. The default is one million bytes.			Client, family server
TC_USER	Specifies the user login ID for single-user environments OS/2, Windows 3.1, and Windows 95. This environment variable is not used in multiuser environments AIX, HP-UX, Solaris, and Windows NT.		User ID	Client, build server
TC_WORKAREA	Specifies the default work area name.	-workarea	Work area	Client, make import tool
TC_WWWPATH	Specifies the path for the HTML helps and image files for Web client.	-workarea	Work area	Client, family server

---

## Setting environment variables

For methods of setting your environment variables, refer to your operating system documentation. For example, for OS/2 you can use the following command to set the TC\_FAMILY environment variable:

```
SET TC_FAMILY=familyName@hostname@portnumber
```



---

## Appendix E. TeamConnection NLS and DBCS considerations

This section describes how to use TeamConnection in situations that require National Language Support (NLS) and Double-Byte Character Sets (DBCS). Several important related issues are described, such as the need for using the same code page among the clients and the server, and how to setup a workstation to handle different locales.

The topics in this section include the following:

- An overview of the NLS and DBCS support provided by TeamConnection, such as use of locale XPG/4 I18N programming model and supported locales and platforms.
- The main characteristics and limitations related to NLS/DBCS, such as the interoperability between clients and server, and special cases for the manipulation of data by TeamConnection.
- The main issues related to installation, administration and runtime, such as directory structure of the installed code, and why you should not change the code page of an existing TeamConnection family.
- Miscellaneous topics, such as how to install locales in AIX.

**Note:** In OS/2 and Windows, the locale support is not provided by the operating system but by TeamConnection during installation.

---

### Overview of TeamConnection NLS and DBCS support

TeamConnection supports the I18N (Internationalization) locale model proposed by XPG/4 (X/Open Portability Guide, issue 4) in which the language and culture sensitive information are not hard coded in the executables; instead, they are provided as system resources by means of a "locale" that the user can specify at run-time.

One of the components of a locale is the code page in which the characters will be handled. For example, in AIX, the default locale is "en\_US" which stands for English in the USA and the associated code page is ISO8859-1, which is different than the default one used for English in OS/2 (code page IBM-850) but similar to the one used for English in Windows (code page MS-1252 Latin 1).

The locale model for XPG/4 establishes several environment variables that can be used for controlling the culture sensitive information. The following table describes these environment variables, their function, and how TeamConnection deals with them.

Locale environment variable	Function	How TeamConnection uses it
LANG	Specifies the installation default locale.	Identifies the path for the message catalogs and other language-sensitive files.
LC_ALL	Overrides the value of other LC_* environment variables.	It is not explicitly exploited by TeamConnection.
LC_COLLATE	Determines the character-collation or string-collation rules.	It is ignored by TeamConnection. (See Note 1).

LC_CTYPE	Determines the character handling rules governing the interpretation of sequences of bytes of text data characters and classification of characters.	It is ignored by TeamConnection.
LC_MESSAGES	Determines the rules governing affirmative and negative responses, and the locale for messages and menus.	It is ignored by TeamConnection.
LC_MONETARY	Determine the rules governing monetary-related formatting.	It is ignored by TeamConnection, because it does not handle this kind of information.
LC_NUMERIC	Determine the rules governing non-monetary numeric formatting.	It is ignored by TeamConnection, because it handles only integer numbers with no separation for thousands.
LC_TIME	Determine the rules governing date and time formatting.	It is ignored by TeamConnection. There is no special processing for the date and time information. (See Note 2).

**Notes:**

1. TeamConnection itself does not perform any sorting of data. Instead, the sorting is performed by the database. Because only the English version of the database is provided, by default the English character collation rules are used. However, there is an exception: the Japanese locale is provided for the database.
2. The date and time in TeamConnection is represented as follows:  
YYYY/mm/dd hh:mm:ss

Where:

- YYYY is the year
- mm is the month
- dd is that day
- hh is the hour
- mm is the minute
- ss is the second

Because 4 digits are used for the year, TeamConnection is compliant with the 2000 Year specifications.

## Supported locales

TeamConnection provides support for both single-byte character set (SBCS) locales and double-byte character set (DBCS) locales.

## SBCS locales

Each row in the following table represents what code pages and locales are compatible across the different platforms. For example, the locale "En\_US" is different from the locale "en\_US" and therefore each locale is explicitly described in a separate entry.

Table 26. SBCS locales supported by TeamConnection

Language and Country	Locale	Code Page OS/2	Code Page Windows NT/95	Code Page AIX 4
English, USA (enu)	En_US	IBM-437, IBM-850	DOS: MS-437, MS-850 (note 1)	IBM-850
English, USA (enu)	en_US	n/a	GUI: MS-1252 (note 1)	ISO8859-1
Portuguese, Portugal (ptb)	Pt_PT	850	MS-850 (note 1)	IBM-850
Portuguese, Brazil (ptb)	pt_BR	n/a	GUI: MS-1252 (note 1)	ISO8859-1

### Notes:

1. When using directly the TeamConnection line commands from a DOS Command Prompt in Windows, the DOS code pages are used (such as MS-850).

In contrast, when using the TeamConnection GUI the MS code pages are used (such as MS-1252).

It is important to emphasize that the locales "Pt\_PT" and "pt\_BR" are different. For example, if you use a TeamConnection family in AIX with the ISO locale pt\_BR (code page ISO8859-1), and a TeamConnection client in OS/2 with the Pt\_PT locale (code page IBM-850), then you will see "code page incompatibility" problems (in which some characters will NOT be shown or will not look OK).

## DBCS locales

Each row in the following table represents what code pages and locales are compatible across the different platforms. For example, the locale "Ja\_JP" is different than the locale "ja\_JP" and therefore each locale is explicitly described in a separate entry.

Table 27. DBCS locales supported by TeamConnection

Language and Country	Locale	Code Page OS/2	Code Page Windows NT/95	Code Page AIX 4
English, USA (enu)	En_US	IBM-437, IBM-850	DOS: MS-437, MS-850 (note 1)	IBM-850
Japanese, Japan (jpn)	Ja_JP	IBM-932 (note 1)	MS-932 = IBM-943	IBM-932
Japanese, Japan (jpn)	ja_JP	n/a	n/a	IBM-eucJP
Korean, Korea (kor)	ko_KR	IBM-949	MS-949 (UHC) = IBM-1363 (note 2)	IBM-eucKR

Table 27. DBCS locales supported by TeamConnection (continued)

Language and Country	Locale	Code Page OS/2	Code Page Windows NT/95	Code Page AIX 4
Simplified Chinese, RPC (chs)	zh_CN	IBM-1381	MS-936 (GBK) = IBM-1386 (note 3)	IBM-eucCN
Traditional Chinese, Taiwan (cht)	Zh_TW	IBM-950 (note 4)	MS-950 (note 4)	BIG5 (note 4)
Traditional Chinese, Taiwan (cht)	zh_TW	n/a	n/a	IBM-eucTW = IBM-964 (note 5)

**Notes:**

1. The Japanese IBM-932, IBM-942 and IBM-943 have very small differences between them, but generally speaking, they are compatible with each other.
2. The Korean code page for Windows NT/95 is called UHC (Unified Hangeul Code).  
The IBM-1363 code page extends IBM-949 by adding missing Hangeul characters with no change of assignments in code points for IBM-949.
3. The code page for Simplified Chinese for Windows NT/95 is called GBK (Guo Biao Kuo).  
The IBM-1386 code page extends IBM-1381 by adding missing Unicode characters with no change of assignments in code points for IBM-1381.
4. The PC code page for Traditional Chinese is called Big-5.
5. The EUC code page for Traditional Chinese (IBM-eucTW) for AIX 4.1 has been enhanced with respect to AIX 3.2, but it keeps the same locale name (zh\_TW). This means that if the user in AIX 4.1 exploits the new characters in the enhanced locale version, there could be compatibility problems when the user uses the old locale version

It is important to emphasize that the locales "Ja\_JP" and "ja\_JP", and "Zh\_TW" and "zh\_TW" are different. For example, if you use a TeamConnection family in AIX with the EUC locale ja\_JP (code page IBM-eucJP), and a TeamConnection client in OS/2 with the Ja\_JP locale (code page IBM-932) then you will see "code page incompatibility" problems (in which some characters will NOT be shown or will not look OK).

## Platforms supported by TeamConnection

The supported platforms by TeamConnection are shown in the following table.

Table 28. Platform support for TeamConnection components

Platform	Server	Client	Build Agent	Build Processor
OS/2 Warp (3.x)	YES	YES	YES	YES
OS/2 Merlin (4.x)	YES	YES	YES	YES
Windows 3.1 (16-bit)	no	GUI	no	no
Windows 95 (32-bit)	no	YES	no	YES

Table 28. Platform support for TeamConnection components (continued)

Platform	Server	Client	Build Agent	Build Processor
Windows NT 3.51 (32-bit)	YES	YES	YES	YES
Windows NT 4.x (32-bit)	YES	YES	YES	YES
MVS	no	no	no	YES
AIX 4	YES	YES	YES	YES
HP-UX 10	YES	YES	YES	YES

**Notes:**

1. YES: for client means: both GUI and line commands
2. GUI: means ONLY the graphical user interface, no line commands
3. no: means that there is no support

## Locales supported by the ObjectStore database

TeamConnection uses the ObjectStore object-oriented database management system (DBMS), which is enabled to handle DBCS, regardless of the locale. The installation of TeamConnection also includes the installation of ObjectStore and only the English locales for ObjectStore are included in the installation images. This means that if there are messages issued by ObjectStore, then these messages will be in English, regardless of the locale of the client or server. :note. In UNIX, the installation images for ObjectStore include also the Japanese locales. If the ObjectStore server is running with the Japanese locale, then the messages from ObjectStore will be in Japanese.

---

## Characteristics and limitations of NLS and DBCS support

### No conversion of code points when exchanging data

The TeamConnection clients and servers do not alter the code points of the data. This means that the data is NOT converted from one code page to another when entered by the user, when stored in the database used by the family or when exchanged between a client and the server. :p. The information about the code page in which the data was entered is not stored with TeamConnection objects; furthermore, there is no exchange of information between the client and the server to indicate which code page is being used by each of them.

### No impact if using English characters

Because most code pages have the same code points for the first 128 characters, which includes all the characters used in the English alphabet, then in practice there is no effect in using different code pages between clients and servers, if using only English characters.

As an example, the default multilingual code page for OS/2 is IBM-850, for Windows is MS-1252 Latin 1, and for AIX Version 4 is ISO8859-1. In these code pages the first 128 characters are the same, and thus, there is no impact in code points the English characters are used when remarks are entered for a defect in the OS/2 client, stored in the AIX server and retrieved by the Windows client.

For example, the code point value of 100 is the lower case letter "d" which has the same graphic representation in most of the code pages, as exemplified in the following table.

*Table 29. Graphical representation of code point 100 in several code pages*

Platform	Locale	Code Page	Representation
OS/2	English	IBM-437	lower case 'd'
OS/2	English	IBM-850	lower case 'd'
Windows, DOS mode	English	MS-437	lower case 'd'
Windows, DOS mode	English	MS-850	lower case 'd'
Windows, Graphical	English	MS-1252	lower case 'd'
AIX	En_US	IBM-850	lower case 'd'
AIX	en_US	ISO8859-1	lower case 'd'
OS/2	Japanese	IBM-932	lower case 'd'
Windows	Japanese	MS-932	lower case 'd'
AIX	ja_JP	IBM-eucJP	lower case 'd'

## Impact if using non-English characters

However, if the customer wants to use non-English characters, which are characters with code points greater than 128, such as accented characters, umlauts, double-byte characters, then the code pages differ greatly in this respect.

For example, the character with code point value of 252 (which can be entered by pressing ALT and typing 2, 5 and 2 from the numeric keypad in most systems) has the following different representations, as shown in the following table.

*Table 30. Graphical representation of code point 252 in several code pages*

Platform	Locale	Code Page	Representation
OS/2	English	IBM-437	superscript 'n'
OS/2	English	IBM-850	superscript '3'
Windows, DOS mode	English	MS-437	superscript 'n'
Windows, DOS mode	English	MS-850	superscript '3'
Windows, Graphical	English	MS-1252	lower case 'u' with dieresis
AIX	En_US	IBM-850	superscript '3'
AIX	en_US	ISO8859-1	lower case 'u' with dieresis
OS/2	Japanese	IBM-932	First byte of DBCS character
Windows	Japanese	MS-932	First byte of DBCS character
AIX	ja_JP	IBM-eucJP	First byte of DBCS character

In the above case, a German customer using Windows in Graphical Mode, with code page MS-1252 may enter a string that contains the u with umlaut and store it

in TeamConnection, but the same customer when retrieving the data from OS/2 using IBM-850 code page, the character in the string will be shown as the number 3 in superscript.

### **To maximize compatibility, use same/similar code page**

As shown in “Impact if using non-English characters” on page 438, it is important that the customers who are using multiple platforms with TeamConnection, must understand the implications of using different code pages when dealing with non-English characters.

If possible, the customer should use the same (or similar) code page in the TeamConnection client and in the server.

### **Once a family is created, do not change the code page**

To avoid compatibility problems, if a family is created and used with a given code page, then this code page should not be changed later on.

For example, if a family is created with the Japanese IBM-932 code page in OS/2 and then migrated to the Japanese IBM-eucJP code page in AIX, then there might be several DBCS characters that are valid in the IBM-932 code page that will not be displayed properly when using the IBM-eucJP code page.

### **Using UNICODE in the future to solve incompatibilities**

In the future, once the support for the UNICODE code page is widespread and available in all the platforms supported by TeamConnection, then the customer could choose to use the UNICODE code page for the clients and the server, and in this way, avoid the current incompatibility between different code pages.

Another alternative that we studied to solve to this incompatibility problem between code pages was to add an extra field for EVERY SINGLE piece of data that is handled by TeamConnection in order to identify the code page that was used when the data was originated; then, this would require that the TeamConnection server should get the code page used by each client that is requesting a service, and then do the necessary conversions when exchanging the data. Because this alternative is very expensive to implement and has a lot of ramifications, and because UNICODE is the right way for the long term, we are not implementing this alternative to tag each piece of data.

## **Exceptions to the handling of characters in TeamConnection**

### **The ! split vertical bar character could be changed**

The !split vertical bar character is used to separate the fields in the “teamc report -raw” command. Thus, if this character is found in a field that is shown by this command, such as in the abstract of a defect, then the character is changed to “!” (exclamation point) by the TeamConnection client. Thus, the server does not see these split vertical bar characters.

The reason for this change is to avoid confusion during the parsing of the -raw output because the split vertical bar is used to separate the fields. If in the output to be parsed there is a split vertical bar character that is NOT intended to be a separator of a field, then the parsing routine will not be able to guess that this particular split vertical bar should not be considered as a field separator. In other



words, ALL split vertical separator bars are considered to be field separators, and thus, any such characters in the abstract will not be parsed appropriately.

For example, when opening a defect, if the abstract field is left blank then the first 63 characters of the remarks field will be placed in the abstract. The abstract is a field that is shown with the "teamc report -raw" command, but the remarks field is not shown with this command. Thus, if the first 63 characters of the remarks have split vertical bar characters they will be left untouched in the actual remarks, but they will be converted to "!" in the abstract.

## Keyword expansion

TeamConnection supports the expansion of certain keywords embedded in the text during the extraction of text files. The routines that handle the expansion are NLS and DBCS enabled.

The important characteristic to remember is that the expansion is done by the TeamConnection family server and not by the client.

## CR (carriage return) and LF (line feed)

Although this is not an NLS issue, this is another topic that is worth including in this technical report, because some users may think, incorrectly, that this could be caused by code conversion processing done by TeamConnection.

The end of a line of text in OS/2 or in Windows is represented by the character pair CR-LF (carriage return and line feed), whereas in UNIX is represented simply by the character LF (line feed).

In TeamConnection, the model of what-you-see-is-what-you-get is used. This means that if a user creates a file in TeamConnection, regardless of the platform of the server, then TeamConnection will NOT do any conversion of LF or CRLF on that file. There are choices in the -extract action to allow for more fine tuning of these on-the-fly-conversions. For example, an AIX user may wish to extract with only LF a file that was stored originally from OS/2 that has CRLF.

The following file will be the source file to be used in the rest of the examples in this section:

```
This is line 1
This is line 2
This is line 3
```

If the source file is created from an OS/2 client and later on is extracted into a UNIX client without CRLF conversion, then the resulting file will have the CR character at the end of each line and the file would look like:

```
This is line 1^M
This is line 2^M
This is line 3^M
```

If the source file is created from a UNIX client and later on is extracted into an OS/2 client without CRLF conversion, then the resulting file will not have the CR character at the end of each line and the file would look like:

```
This is line 1
          This is line 2
                This is line 3
```



## All clients in the same host must use the same language (Intel only)

For OS/2 and Windows NT clients, all the TeamConnection clients that execute from one single host must use the same language if they run at the same time.

This limitation is due to the inherent limitation of these platforms in which ONLY ONE version of given DLL can be loaded at the same time, and because these platforms are not fully compliant with the XPG/4 model that allows usage of multiple locales. If there are different versions of some DLLs for each language, and if the English version of the DLL is loaded, the Japanese one cannot be loaded at the same time. This precludes having clients that have different languages to run at the same time.

## Untranslated strings that are visible to the users

There are certain kinds of strings that are visible to the user that are not translatable:

- command, action and flag names
- state names
- database table and view names
- database table column headings
- action name used in the audit log, the mail notifications, and the authority and interest tables
- the type field in the config database table

## DBCS Limitations

The following limitations apply to DBCS character sets:

1. The administration tools for the TeamConnection Server expect SBCS characters as the reply for Yes (y) and No (n).
2. The administration tools for the TeamConnection Server have the following limitations for DBCS:
  - a. The \*.ld files (authority, interest, cfgcomproc and cfgrelproc) in the family account can accept DBCS characters in the first field for each entry. The maximum size for this field is 15 bytes.
  - b. The config.ld file in the family account can accept DBCS characters in the following fields (the positions are defined from left to right):
    - Field position 1 ("Field Type"): limit is 15 bytes
    - Field position 2 ("Value"): limit is 15 bytes
    - Field position 6 ("Description"): limit is 63 bytes
  - c. The chfield program can accept only SBCS characters in the following fields:
    - CMD attribute
    - DB Column Name
  - d. The chfield program can accept DBCS characters in the following fields:
    - Field label: limit is 15 bytes
    - Title label: limit is 15 bytes
    - Type: must be a valid type defined in config.ld (limit is 15 bytes).
3. The TeamConnection Commands Reference manual, in Appendix A, "Querying the TeamConnection database", shows the datatype and the size limit for the attributes of the TeamConnection objects; however, the actual size limit for

many of the character attributes is smaller than the specified limit. For example, the field "login" in the "Users" table shows that the limit is 31 bytes, but in reality only 15 characters (SBCS or DBCS) can be stored in that field. The fields affected are usually related to names, such as the User login, the Component name, etc.

If you specify a string that has DBCS characters and that the size of the string goes beyond the limit, then the following error message will be displayed by the TeamConnection server:

0010-149 Your request cannot be completed. The attribute flag argument xxx is not valid.

4. Warning on the use of 0x7C as a second byte in a DBCS character

The 0x7C character corresponds to the vertical bar ('|') which in TeamConnection is interpreted as a field separator when dealing with reports and with handling windows and fields in the GUI.

You can use this value as the 2nd byte of a DBCS character, however, when the data that contains this 2nd byte is handled in a TeamConnection client that has an SBCS code page (and not a DBCS code page), then, the output shown by the client may be displaced, that is, the 0x7C value will be interpreted as the field separator. Moreover, this situation will apply for any string in the \*.ld files and in the configurable fields.

---

## Installation, administration and runtime issues

### Installation issues related to NLS and DBCS

The installation process for TeamConnection is similar in UNIX, in OS/2 and Windows with respect to NLS. The similarities and the differences are explained in the following sections.

After the installation process, the executable code and the language related files will be installed in separate directories that are system wide, that is, they are not exclusive to one account.

When a TeamConnection family is created, several files are copied into the directory for the TeamConnection family; several of these files contain language sensitive information (such as the config.ld file and the files in the chfField directory). The family administrator can modify these files for the specific family; these files are not shared with other families.

#### Using a similar directory structure across all the platforms

Even though there are differences in the NLS facilities that are available from the UNIX and the Intel (OS/2 and Windows) platforms the installation of TeamConnection in these platforms creates a similar directory structure whose top directory is shown below (using the default directory):

**AIX 4** /usr/teamc

**HP-UX 10**  
/opt/teamc

**OS/2** c:\teamc

**Windows 3.1**  
c:\ProgramF\TeamC

**Windows NT and 95**  
c:\Program Files\TeamConnection

**Storing the language-independent files:** The language-independent files for the TeamConnection code are stored in similar directories, as shown in the following example. The teamc server daemon (teamcd) is located in the subdirectory "bin", from the TeamConnection top directory as shown below:

**AIX 4** /usr/teamc/bin

**HP-UX 10**

/opt/teamc/bin

**OS/2** c:\teamc\bin

**Windows 3.1**

c:\ProgramF\TeamC\bin

**Windows NT and 95**

c:\Program Files\TeamConnection\bin

**Storing the language-dependent files:** In a similar way, the language-dependent files for the TeamConnection code are stored in a similar subdirectory structure, which is the subdirectory "nls" as parent and then the "msg" for messages and "cfg" for configuration items.

For example, the ISO US English message catalog will be stored as shown below, using the default location:

**AIX 4** /usr/teamc/nls/msg/en\_US (which really is a symbolic link to /usr/lib/nls/msg/en\_US)

**HP-UX 10**

/opt/teamc/nls/msg/C (which really is a symbolic link to /usr/lib/nls/msg/C)

**OS/2** c:\teamc\nls\msg\en\_US

**Windows 3.1**

c:\ProgramF\TeamC\nls\msg\en\_US

**Windows NT and 95**

c:\Program Files\TeamConnection\nls\msg\en\_US

**List of language-dependent files:** The "nls" directory (see previous section for the complete path) contains the following subdirectories and files:

**nls/msg//**

All message catalog files, such as teamcv2.cat; all help files; all resource DLLs for the GUI that are specific to a language.

**nls/doc//**

All documentation: PostScript, HTML, online, etc.

**nls/cfg//**

All configuration files, such as config.ld, and files for the configurable fields; the original teamc20.ini file.

## Installation issues for UNIX

During the installation process for TeamConnection in UNIX, it is necessary to select the appropriate language version to install. The code is not bundled together with the language sensitive information. That is, there is an individual installable package just for the language sensitive information that could be installed independently.

Because AIX 4 and HP-UX 10 operating systems already include the explicit support for the XPG/4 I18N locale model, the TeamConnection installation process will not install additional files for this matter (as in OS/2 and Windows).

The message catalog that contains the language sensitive information is located by the executable code by means of the combination of the NLSPATH and LANG environment variable. By default, this variable is set as follows:

```
set NLSPATH=/usr/lib/nls/msg/%L/%N
```

Where:

- %L is a variable that at runtime represents the value of the LANG environment variable; it must be in uppercase.
- %N is a variable that at runtime represents the name of the message catalog to be used; it must be in uppercase.

## Installation issues with OS/2 and Windows

During the installation process for TeamConnection in OS/2 and Windows, it is necessary to select the appropriate language version to install. The code and the language sensitive information is bundled together in a package and it is installed appropriately. That is, there is not an individual package just for the language sensitive information that can be installed independently.

Because the OS/2 and Windows operating systems do not include at this moment explicit support for the XPG/4 I18N locale model, the TeamConnection installation process will install any necessary support for this model.

The message catalog that contains the language sensitive information is located by the executable code by means of the NLSPATH environment variable. By default, this variable is set as follows:

```
set NLSPATH=:\teamc\nls\%N
```

Where:

- :\teamc represents the appropriate drive and top directory where the TeamConnection code is installed in your system
- nls is the directory that contains the NLS related files
- %N is a variable that at runtime represents the name of the message catalog to be used; it must be in uppercase.

## Family administration issues

### A family should use the same language all the time

Although technically it could be possible for a family to be created using the en\_US locale and then change it later on to another language, we consider that this process has the potential to cause a lot of confusion with the users, especially for the mapping of code points.

Therefore, this is treated as a limitation and if the customers try it, it is at their own risk and we will not help them. However, the customer may decide to delete the family, change the language by reinstalling the code for Intel and specify the new language, or to install the new language message catalogs for UNIX and change the LANG variable, and then create a new family to use the new setting.

This decision affects the arrangement of the subdirectories of a family: there is no provision in either UNIX or Intel to have language dependent directories inside the family directory.

The following sections contain examples that will clarify this point.

**UNIX:** An AIX customer installed the TeamConnection server, using the en\_US locale. The config.ld file (which is language dependent) resides in /usr/lib/nls/cfg/en\_US/config.ld. The "testfam" TeamConnection family is created, and the config.ld file is copied from the system directory to the top directory of the family, /home/testfam/config.ld. There is no "/home/testfam/en\_US/config.ld" path.

**Intel:** An OS/2 customer installed the TeamConnection server, using the en\_US locale. The config.ld file (which is language dependent) resides in c:\teamc\nls\cfg\en\_US\config.ld. The "testfam" TeamConnection family is created, and the config.ld file is copied from the system directory mentioned above into the top directory of the family, c:\testfam\config.ld. Notice that there is no "c:\testfam\en\_US\config.ld" path.

## Client runtime issues

### A client should use the same language all the time

Although technically it could be possible for a TeamConnection client to be installed with one language (such as the IBM-850 code page in OS/2 or the en\_US locale in AIX) and then change the language in the middle, this process has the potential to cause a lot of confusion with the users, specially for the mapping of code points with the teamc20.ini file, as explained below.

In the UNIX platforms, thanks to the use of the LANG variable, it would be possible to install additional message catalogs for other languages and the user could setup the language to use by setting the variable LANG. However, the teamc20.ini file for the GUI will NOT be changed, and this file may contain characters that were valid in the original setup but that cannot be displayed in the new setup.

Because the Intel platforms do not provide the LANG variable, then it is not possible to have message catalogs for multiple languages for TeamConnection. This means that if the customer decides to change the language then it is necessary to reinstall the code specifying the new language.

The following sections contain examples that will clarify this point.

**UNIX:** A Japanese AIX customer installs the TeamConnection client using the ja\_JP and en\_US message catalogs. The teamc20.ini files (which are language dependent) reside in /usr/lib/nls/cfg/ja\_JP/teamc20.ini and /usr/lib/nls/cfg/en\_US/teamc20.ini. The customer uses the Japanese GUI for the first time (LANG=ja\_JP), and the GUI detects that the following file does not exist: \$HOME/teamc20.ini. The customer uses the GUI and creates several entries written in Japanese in the task list which are stored in the teamc20.ini file. The customer exits the GUI. The user invokes the GUI again, and the GUI detects that the teamc20.ini file exists in \$HOME and therefore the GUI uses it, and does not try to overwrite it with the file in the directory /usr/lib/nls/cfg/ja\_JP. The customer decides then to switch the locale to en\_US, by setting LANG=en\_US, exits and logs in again.

The user brings up the English TeamConnection GUI and now the task list shows entries that may not be legible because their original code points were set with the ja\_JP locale.

**Intel:** A Japanese OS/2 customer installs the TeamConnection client and specifies the ja\_JP language only, because she cannot install multiple languages. The code page is IBM-932. The PATH and the NLSPATH variables point to the ja\_JP directories. The teamc20.ini file (which is language dependent) resides in c:\teamc\nls\cfg\ja\_JP\teamc20.ini. The customer uses the Japanese GUI for the first time (LANG=ja\_JP), and the GUI detects that the following file does not exist: c:\os2\teamc20.ini. The GUI copies the original teamc20.ini file from the appropriate ja\_JP directory, c:\teamc\nls\cfg\ja\_JP\teamc20.ini, into c:\os2\teamc20.ini. The customer uses the GUI and creates several entries written in Japanese in the task list, and this list is stored in the teamc20.ini file. The customer exits the GUI. The user invokes the GUI again, and the GUI detects that the teamc20.ini file exists in c:\os2 and therefore the GUI uses it, and does not try to overwrite it with the file in the directory c:\teamc\nls\cfg\ja\_JP. The customer decides then to switch the locale to en\_US, by uninstalling the TeamConnection client and reinstalling it again specifying now the language en\_US, and reboots. The PATH and the NLSPATH variables are updated and they do not point to the non-existing ja\_JP directories, but point to the new en\_US directories. If the customer keeps the same code page, IBM-932, then when the user brings up the English TeamConnection GUI, the task list shows entries that are legible because their original code points were set with the IBM-932 code page. If the customer changes the code page, let's say to IBM-850, then when the user brings up the English TeamConnection GUI the task list shows entries that may not be legible because their original code points were set with the IBM-932 code page.

---

## Miscellaneous topics that are specific to operating systems

### Problem resolution

#### Potential problems if the language locale is not set properly

If the language locale is not set properly, it can cause the following problems:

- Unexpected message catalogs would be picked up.
- GUI labels, titles, and messages could be unreadable.
- Data conversion errors could be resulted.
- 

**Note:** Default hard coded messages are in English.

If any of the above symptoms occur, verify the locale and reset appropriately.

#### When using LANG=C in AIX, the message catalogs are ignored!

The IBM red book "AIX 3.2 National Language Support", GG24-3850, says on page 16: "when the default C locale is the current locale, the hardcoded default message catalogs in the executable code will be used instead of the translated message file." The main point for confusion is that there is a directory in /usr/lib/nls/msg/C which is totally ignored! Thus, the user can place the message catalogs there, but they will never be used! This means that if the TeamConnection Server is using the LANG=C locale, then the output for "teamc report -testServer" will always be that the message catalog is not available and thus, the server will use the internal English messages which are embedded during compilation and used as default.

## Using misc/tcmsgcat to verify the NLS message catalog settings

The utility "misc/tcmsgcat.aix" provided in the CD-ROM for TeamConnection can be used to verify that all the NLS variables related to the message catalog for TeamConnection are properly configured and that the message catalog itself can be opened; furthermore, the first message in the message catalog is read, and it contains the name of the language used with the strings in the message catalog.

The following is sample output with the English en\_US message catalog:

```
Testing for the existence of the message catalog: teamcv2.cat
The value of the LANG variable is:          en_US
The value of the LC_MESSAGES variable is: not explicitly defined
The value of the NLSPATH variable is: /usr/lib/nls/msg/%L/%N
SUCCESS! the message catalog teamcv2.cat was opened.
Printing from the catalog, the message in position: 1
English
You can use 'dspmsg msg.cat -s 1 ' to display other messages
Example, the first message in teamcv2.cat does not require arguments:
    dspmsg teamcv2.cat -s 1 1
Example, the second message requires 5 arguments (add dummy s0):
    dspmsg teamcv2.cat -s 1 2 s0 s1 s2 s3 s4 s5
```

Actually, you can use the TeamConnection utility "showmsg" which is a front end to the AIX utility "dspmsg"; for more information see "Using misc/showmsg to display a given message from the catalog".

## Using misc/showmsg to display a given message from the catalog

The utility "misc/showmsg" provided in the CD-ROM for TeamConnection can be used to display a particular message from the TeamConnection message catalog.

This utility is actually more helpful for our development team, because in order to display a given message we need to find out the actual number to use with the utility showmsg. The utility uses dummy positional input variables which will be replaced, if needed, when displaying the message.

If we want to display the message "0011-209" from the message catalog, we need to identify which is the mnemonic and the index for the message, which can be obtained from the file src/inc/tcmsgdef.hpp which is generated during the construction of the message catalog. For example, the search for "0011-209" gives the following entry:

```
... "msgLicenseMonitorInvalidDateRange","0011-209",2530);
```

Then, we use the index 2530 with the utility "showmsg" to verify if indeed we obtain the text for the message 0011-209, as shown below.

```
message 2530 from message catalog:
/usr/lib/nls/msg/en_US/teamcv2.cat
0011-209 The date range from s1 to s2
        is not valid. The end date must be greater than the begin date.
```

## Problems when the UNIX teamcgui window does not show up

When using UNIX, if you start the GUI by using "teamcgui" or "teamcgui &" (to send the process to the background), it takes several seconds for the GUI to actually appear in your display.



However, if after several minutes of waiting and still you do not see a window for the TeamConnection window, you can kill the runaway process:

```
ps -ef | grep teamcgui
Identify the process id for teamcgui
kill -15 processIdforTeamcgui
```

Then check the following:

- Ensure that the DISPLAY variable is properly set to your X server.
- The teamc20.ini file might be corrupted. You can delete it or rename it, and start again teamcgui.

## Topics related to the message catalog used by TeamConnection

### Using the "teamc report" command to test the NLS settings

You can use the commands to verify if TeamConnection is accessing the message catalog:

- For the server: "teamc report -testServer"

A sample output is shown below.

```
Connect to Family Name:          tcocto
Server TCP/IP Name:             carcps22.raleigh.ibm.com
Server IP Address:              9.67.238.38
Server TCP/IP Port Number:      1300
Server Specific Information -----
Product Version:                2.0.4
Operating System:               AIX
Message catalog language:       English
Server Mode:                    non-maintenance
Authentication Level:           HOST_ONLY
```

- For the client: "teamc report -testClient"

A sample output is shown below

```
Product Version:                2.0.5
Message catalog language:       English
Environment variables:
Variable:                       Setting:
-----
TC_USER                         rivera
TC_BECOME                       rivera
TC_FAMILY                       tcocto@carcps22@1300
TC_RELEASE                      v205
TC_TOP
TC_WORKAREA
TC_BUILDPOOL
```

### How to know when TeamConnection is not using the message catalog

If the output of the commands "teamc report -testServer" or "teamc report -testClient" produce an output line that is similar to the following:

```
Message catalog language:       English (Internal)
```

This means that TeamConnection cannot find the message catalog and is using the internal default English messages that are embedded in the executables during compilation. The most likely cause for this problem is an improper definition of the NLSPATH environment variable.



## Setting and verifying the current locale in UNIX

To set the language locale to something different from the current setting, issue the following depending on what SHELL environment you are in:

- Korn Shell  
`export LANG=Ja_JP (or) export LANG=ja_JP`
- To verify the current locale setting, issue the following command:  
`locale`

The output of the command looks something similar to the following:

```
LANG=Ja_JP
LC_COLLATE="Ja_JP"
LC_CTYPE="Ja_JP"
LC_MONETARY="Ja_JP"
LC_NUMERIC="Ja_JP"
LC_TIME="Ja_JP"
LC_MESSAGES="Ja_JP"
LC_ALL=
```

## How to enter SBCS extended characters using an English keyboard

It is possible to enter any SBCS character defined in the current code page, including characters that are not available from the layout of your keyboard in the following environments:

- In OS/2 windows.
- In the DOS command prompt in Windows.
- In AIX "aixterm".
- In AIX "dtterm" for non-English locales.

If using English locales, you cannot enter these extended characters, but if using Ja\_JP for example, you can. For example, the US English keyboard does not have accented characters, but it is possible to enter them by means of the following procedure:

1. Ensure that the "Num Lock" mode is activated in the numeric keypad, which is located in the right side of the standard keyboard. If your keyboard has a LED indicator for this mode, then it should be on.
2. Press and hold the Alt key.
3. Enter the 3 digits that correspond to the code point in decimal, such as 252: entering the digits 2, followed by 5 and 2.
4. Release the Alt key.

This action can be summarized as "Alt-XXX", such as "Alt-252" for the above example. Of course, if you have a keyboard that can enter extended characters, then you do not have to use the Alt-XXX method.

## How to specify NLS settings for AIX GUI on aixterm/dtterm windows

If you are going to use the TeamConnection line commands and the TeamConnection GUI in Motif in AIX and using different locales, then you may need to open a new window with the aixterm or dtterm (which is provided by the Common Desktop Environment) utilities.

Although these utilities provide a similar function, they differ considerably with respect to the NLS support.

## Using aixterm

1. Open an X-window session with aixterm with the desired locale (code-page) and the appropriate font. The -T "string" parameter for aixterm will determine the title for the window.

Some examples are shown below:

**For IBM-850:**

```
aixterm -T "IBM-850" -lang En_US -fn Rom14 &
```

**For ISO-8859-1:**

```
aixterm -T "ISO-8859-1" -lang en_US -fn Rom14.iso1 &
```

**For Ja\_JP:**

```
aixterm -T "Ja_JP" -lang Ja_JP -fn *gothic*-19* &
```

**For ja\_JP:**

```
aixterm -T "ja_JP" -lang ja_JP -fn *gothic*-19* &
```

2. Perform the quick test for the code page, by entering Alt-252. If code page IBM-850, then you will see 3-superscript; if code page ISO-8859-1, you will see u-umlaut.
3. For DBCS, ensure that the aixterm window has the keyboard status area located at the bottom left corner of the window; also, ensure that you can enter SBCS and DBCS characters.

## Using dtterm

1. Open an X-window session with dtterm with the desired locale (code-page) and the appropriate font. The -name "string" parameter for dtterm will determine the title for the window.

Some examples are shown below:

**For IBM-850:**

```
LANG=En_US dtterm -name "IBM-850" -fn Rom14 &
```

**For ISO-8859-1:**

```
LANG=en_US dtterm -name "ISO-8859-1" -fn Rom14.iso1 &
```

**For Ja\_JP:**

```
LANG=Ja_JP dtterm -name "Ja_JP" -fn *gothic*-19* &
```

**For ja\_JP:**

```
LANG=ja_JP dtterm -name "ja_JP" -fn *gothic*-19* &
```

2. Perform the quick test for the code page, by entering Alt-252. If code page IBM-850, then you will see 3-superscript; if code page ISO-8859-1, you will see u-umlaut.
3. For DBCS, ensure that the aixterm window has the keyboard status area located at the bottom left corner of the window; also, ensure that you can enter SBCS and DBCS characters.

## Creating a Teamcgui resource file for AIX

You can create a resource file for the TeamConnection GUI for AIX, called "Teamcgui" (the first T of this file name is in uppercase, and the rest of the string is in lowercase) in your home directory to specify the appropriate font.

For example, for Japanese, you can specify the following small size font:

```
*fontList:      *gothic--19*:
*XmText*fontList: *gothic--19*:
*XmList*fontList: *gothic--19*:
```

You could use these other fonts for Japanese:

```
Small font:  *gothic--19*:
Medium font: *mincho--27*:
Large font:  *mincho--35*:
```

**Note:** You must specify the colon at the end of the font specification; if the colon is not present, such as in `*gothic--19*` instead of `*gothic--19*`: otherwise the keyboard status area will not be shown in the bottom left corner of the TeamConnection GUI.

## Using iconv to convert files to other code pages

The source files for the message catalog and other source files such as `config.ld` for TeamConnection were created in US English, using code page IBM-850 in OS/2. Then these files were sent to the translation centers and the translation was done using the corresponding code pages for OS/2. This means that in order to support the code pages in UNIX, we needed to convert the code pages for certain files. This conversion can be accomplished by installing the appropriate locales and using the utility "iconv", as shown below:

### From EN\_US to en\_US

```
iconv -f IBM-850 -t ISO8859-1 input-file > output-file
```

### From Ja\_JP to ja\_JP

```
iconv -f IBM-932 -t IBM-eucJP input-file > output-file
```

### For ko\_KR

not needed

### From Pt\_PT to pt\_BR

```
iconv -f IBM-850 -t ISO8859-1 input-file > output-file
```

### For zh\_CN

not needed

### From Big-5 to zh\_TW

```
conv -f big5 -t IBM-eucTW input-file > output-file
```

The iconv utility uses the names supplied in the "from" and "to" input parameters to obtain the name of the converter file to be used for the conversion, by concatenating the fields and inserting an underscore character between them. In the following example, the converter file is "IBM-850\_ISO8859-1":

```
iconv -f IBM-850 -t ISO8859-1 input-file > output-file
```

The converter files are provided with the file sets used to add a new cultural convention to your host; the converter files are stored in `/usr/lib/nls/iconv`. Thus, if you want to know if your host has a converter file from one code page to another, you can go to this directory and do a file search of the converter file names that have the desired code page as a substring. For example, to find out all the converters related to the IBM-850 code page you can issue the following commands:

```
cd /usr/lib/nls/iconv
ls *850*
```

## How to install additional locales for AIX

Do the following to install additional locales for AIX:

1. Insert the appropriate CD-ROM with the installation images for the AIX operating system.
2. Login as root.
3. Invoke smit (or smitty).
4. Select System Environments.
5. Select Manage Language Environment.
6. Select Manage Language Environment.
7. Select Add Additional Language Environments.
8. Select CULTURAL convention to install

This item includes the locale information and the conversion files between code pages related to the locale.

User F4 to list the choices from the CD-ROM. The following entries were installed in our build and testing machines:

```
IBM-850      English (USA) [En_US]
ISO8859-1    English (USA) [en_US]
IBM-eucCN    Chinese (Simplified EUC) [zh_CN]
IBM-eucTW    Chinese (Traditional) [zh_TW]
IBM-eucJP    Japanese (EUC) [ja_JP]
IBM-932      Japanese (PC) [Ja_JP]
IBM-eucKR    Korean [ko_KR]
ISO8859-1    Portuguese (Brazil) [pt_BR]
```

9. Select LANGUAGE translation to install.

This item includes the message catalogs that will be stored in /usr/lib/nls/msg.

:Use F4 to list the choices from the CD-ROM.

10. Press enter to install the desired locales and/or message catalogs.
11. Press F10 to exit from smit.

## How to change the keyboard for an AIX host

**Note:** You MUST use the RISC keyboard and NOT a PC keyboard. If you use a PC keyboard then you cannot enter any characters after rebooting the machine. To reset the keyboard setting you will need to change the keyboard to a RISC keyboard, then telnet from another host to reconfigure smit, shutdown, and reboot).

1. Insert the appropriate CD-ROM with the installation images for the AIX operating system.
2. Login as root
3. Invoke smit (or smitty).
4. Select: System Environments
5. Select: Manage Language Environment
6. Select: Change/Show Primary Language Environment
7. Select: Change/Show Cultural Convention, Language, or Keyboard
8. Enter the desired values for the following items:
  - Primary CULTURAL convention
  - Primary LANGUAGE translation
  - Primary KEYBOARD

Use F4 to list the choices from the CD-ROM. The choices must match, otherwise there will be an error. For example, if installing a Japanese keyboard but keeping the English cultural convention, then this combination does not match and will not succeed.

9. Press enter to make the change.
10. Exit smit, by pressing F10.
11. Shutdown the machine and DO NOT REBOOT yet:  
`shutdown -F`
12. Change the physical keyboard.
13. Boot the machine.
14. Verify that you can use the new keyboard.

## How to install a non-English PC keyboard to an X-station

The AIX hosts that run on RISC boxes require keyboards that are not compatible with the PC standard keyboards. If you have a Japanese keyboard for a PC, for example, you cannot use this keyboard with a RISC machine. That is, you must get a Japanese keyboard that is specifically designed for the RISC machine. However, it is possible to connect a PC keyboard to an X-station which can communicate with a RISC server. Follow these instructions to accomplish this task. The PC keyboard (where: is the supported language) C Posix Language, and C Posix cultural convention seems to be the most versatile for using different keyboard definitions from X-stations. This convention can be set via smitty:

1. smitty
2. Devices
3. Xstation Configuration
4. Add an Xstation
5. Select the desired model
6. From the "Add an Xstation" window, specify the following values that are related to NLS settings. then press enter to activate.

### **CULTURAL convention**

The desired locale.

### **LANGUAGE**

ISO8859-1 C (POSIX) [C]

### **KEYBOARD**

ISO8859-1 C (POSIX) keyboard [C]

7. Exit smitty
8. Once the correct keyboard definition is configured for the X-station, the change of cultural convention and language settings are as simple as changing the LC\_ALL and LOCALE environment variables.
9. Try not to use the dtlogin panel from X-stations.

When configuring the X-station do not use the /etc/dt/config instructions from x\_sta\_mgr install for CDE. Use the default x session to bring up a dtsession from /usr/dt/bin/dtsession instead, after setting the desired language environment variables.

This is necessary because dt has 2 required fixes for dt to work in an X-station configuration from the default system. These fixes are not guaranteed to fix all the problems, though.

## How to specify a different keyboard mapping

Let's suppose that you have installed the Ja\_JP locale and you only have an English keyboard. You can still enter Japanese characters using the English keyboard by doing the following:

1. `export LANG=Ja_JP`
2. `xmodmap /usr/lpp/X11/defaults/xmodmap/$LANG/keyboard`
3. You may need to find out what is the actual keyboard mapping in order to activate the different input modes and to enter Japanese characters.

## How to start/shutdown the Xstation environment

In case you have a PC keyboard attached to an Xstation environment, you may need to follow the procedures mentioned below to start and stop the Xstation environment.

### How to start the Xstation environment

To start the Xstation environment, do the following:

1. Turn on the Xstation and wait for the login prompt.
2. At the login prompt accept the default host name. Enter the appropriate user id.  
At this point, you are logged in to the system but you do not have still an Xwindow environment.
3. Create an additional profile that contains the desired LANG and keyboard mapping. For example, for using the zh\_TW locale, the profile could be called "profile.zh\_TW". Actually, because it is difficult to enter the underscore character from DBCS keyboards, it is better to not include the underscore in the file name, such as "profile.zhTW". Do not touch the original .profile.
4. Execute the desired profile, such as `../profile.zhTW`.
5. Start the dtsession, which will use the current LANG variable to determine the locale to be used, `/usr/dt/bin/dtsession`.

**Note:** Do not start the dtsession in the background; that is, do not add the & at the end of the statement. Otherwise, it will be more difficult to shutdown the Xwindow environment. Wait several minutes for the Xwindow environment to initialize.

6. Now you can open more windows and start TeamConnection tasks, such as starting the family server or the GUI.

### How to shutdown the Xstation environment

To shut down the Xstation environment, do the following:

1. Go to the window login that started the session.
2. Press ctrl-C to stop the session.
3. A window will display a message asking you to confirm if you want to close the Xwindow environment. The message might be displayed in the language used in the LANG variable. To answer "yes" click on the left button of the dialog. At this point the dtsession will be terminated but the initial windowless login will remain.
4. If you want to restart the Xwindow environment with another locale, simply invoke the appropriate profile and invoke again dtsession.

5. If you want to reboot the Xwindow environment, then press the following keys at the same time: Ctlr-Alt-Backspace.

## **How to start the desktop environment to show the DBCS titles**

In order to display the DBCS titles in any window in your desktop environment, it is necessary that the LANG environment variable of the user that initiates the desktop environment must be of a locale that includes DBCS processing. During our system testing, the user login used to start the X Window desktop environment had a LANG=en\_US and then an aixterm or dtterm window was spawned with the appropriate LANG variable to handle the DBCS locales. In this scenario everything went fine EXCEPT that the titles of the TeamConnection windows that included DBCS characters were NOT shown.





---

## Appendix F. Authority and notification for TeamConnection actions

TeamConnection ships with IBM-supplied authority groups, interest groups, component processes, and release processes. Your family administrator can modify these preconfigured authority groups, interest groups, and processes to fit the needs of your organization.

Each authority group consists of actions normally performed by a particular type of user. Your family administrator can modify these groups or create new ones to reflect the needs of your organization.

Authority groups provide explicit authority to perform the actions included in each group. You might also have implicit authority to perform certain actions according to the objects that you own. Authority groups are defined in a file called `authorit.Id`.

To determine your authority groups, from the Actions pull-down menu, select Lists → Access lists → Show authority actions. On the Show authority actions window select an action.

Each notification group consists of actions normally of interest to a particular type of user. Your family administrator can modify these groups or create new ones to reflect the needs of your organization. Interest groups are defined in a file called `interest.Id`.

To determine your interest notification groups, from the Actions pull-down menu, select Lists → Notification lists → Show interest actions. On the Show authority actions window select an action.

The following table lists all of the TeamConnection actions, the required level of implicit and explicit authority to perform the action, and the users who are notified when an action is performed. To explicitly assign authority to a user, add the user's ID to a component's access list.

**Note:** The user who performs the action is excluded from the notification that is sent out after the action is successfully completed.

For this action	These users have authority	These users are notified
AccessCreate	<ul style="list-style-type: none"><li>• Component owner</li><li>• Explicitly defined for the component where access is being added</li></ul>	User being given new access, subscribers
AccessDelete	<ul style="list-style-type: none"><li>• Component owner</li><li>• Explicitly defined for the component where access is being altered</li></ul>	User whose access was deleted, subscribers
AccessRestrict	<ul style="list-style-type: none"><li>• Component owner</li><li>• Explicitly defined for the component where access is being restricted</li></ul>	User whose access is being restricted, subscribers

<b>For this action</b>	<b>These users have authority</b>	<b>These users are notified</b>
ApprovalAbstain	<ul style="list-style-type: none"> <li>Approval record owner</li> <li>Explicitly defined for the component that manages the associated release</li> </ul>	Approval record owner, subscribers
ApprovalAccept	<ul style="list-style-type: none"> <li>Approval record owner that manages the associated release</li> </ul>	Approval record owner, subscribers
ApprovalAssign	<ul style="list-style-type: none"> <li>Approval record owner</li> <li>Explicitly defined for the component that manages the associated release</li> </ul>	New and original approval record owners, subscribers
ApprovalCreate	<ul style="list-style-type: none"> <li>Work area owner</li> <li>Explicitly defined for the component that manages the associated release</li> </ul>	New approval record owner, subscribers
ApprovalDelete	<ul style="list-style-type: none"> <li>Explicitly defined for the component that manages the associated release</li> </ul>	Approval record owner, subscribers
ApprovalReject	<ul style="list-style-type: none"> <li>Approval record owner</li> <li>Explicitly defined for the component that manages the associated release</li> </ul>	Approval record owner, subscribers
ApproverCreate	<ul style="list-style-type: none"> <li>Release owner</li> <li>Explicitly defined for the component that manages the associated release</li> </ul>	New approver, subscribers
ApproverDelete	<ul style="list-style-type: none"> <li>Release owner</li> <li>Explicitly defined for the component that manages the associated release</li> </ul>	Deleted approver, subscribers
BuilderCreate	<ul style="list-style-type: none"> <li>Explicitly defined for the component that manages the associated release</li> </ul>	Subscribers
BuilderDelete	<ul style="list-style-type: none"> <li>Explicitly defined for the component that manages the associated release</li> </ul>	Subscribers
BuilderExtract	<ul style="list-style-type: none"> <li>Explicitly defined for the component that manages the associated release</li> </ul>	Not applicable
BuilderModify	<ul style="list-style-type: none"> <li>Explicitly defined for the component that manages the associated release</li> </ul>	Subscribers

For this action	These users have authority	These users are notified
BuilderView	<ul style="list-style-type: none"> <li>Explicitly defined for the component that manages the associated release</li> </ul>	Not applicable
CollisionAccept	<ul style="list-style-type: none"> <li>Component owner</li> <li>Explicitly defined for the component that manages the associated release</li> </ul>	Release owner, subscribers
CollisionReconc	<ul style="list-style-type: none"> <li>Component owner</li> <li>Explicitly defined for the component that manages the associated release</li> </ul>	Release owner, subscribers
CollisionReject	<ul style="list-style-type: none"> <li>Component owner</li> <li>Explicitly defined for the component that manages the associated release</li> </ul>	Release owner, subscribers
CompCreate	<ul style="list-style-type: none"> <li>Parent component owner</li> <li>Explicitly defined for the parent component</li> </ul>	New component owner
CompDelete	<ul style="list-style-type: none"> <li>Component owner</li> <li>Explicitly defined for the component being removed</li> </ul>	Component owner, subscribers
CompLink	<ul style="list-style-type: none"> <li>Component owner of the component being linked</li> <li>Explicitly defined for the component being linked</li> </ul>	Owners of both components, subscribers
CompModify	<ul style="list-style-type: none"> <li>Component owner</li> <li>Explicitly defined for the component being modified</li> </ul>	New component owner if applicable, subscribers
CompRecreate	<ul style="list-style-type: none"> <li>Parent component owner</li> <li>Explicitly defined for the parent component</li> </ul>	Owners of both components, subscribers
CompUnlink	<ul style="list-style-type: none"> <li>Component owner of the component being unlinked</li> <li>Explicitly defined for the component being unlinked</li> </ul>	Owners of both components, subscribers
CompView	<ul style="list-style-type: none"> <li>Component owner</li> <li>Explicitly defined for the component being viewed</li> </ul>	Not applicable
CoreqCreate	<ul style="list-style-type: none"> <li>Work area owner of all specified work areas</li> <li>Explicitly defined for the component managing the associated work area and release</li> </ul>	Not applicable

For this action	These users have authority	These users are notified
CoreqDelete	<ul style="list-style-type: none"> <li>• Work area owner of all specified work areas</li> <li>• Explicitly defined for the component associated with the release</li> </ul>	Not applicable
DefectAccept	<ul style="list-style-type: none"> <li>• Defect owner for the component associated with the defect</li> </ul>	Defect owner, defect originator, duplicate defect originators, subscribers
DefectAssign	<ul style="list-style-type: none"> <li>• Defect owner, defect originator</li> <li>• Explicitly defined for the component associated with the defect</li> </ul> <p><b>Note:</b> Originators who do not have DefectAssign authority can reassign the defect only when it is in the open state.</p>	New owner, defect originator, duplicate defect originators, subscribers
DefectCancel	<ul style="list-style-type: none"> <li>• Defect originator</li> <li>• Explicitly defined for the component associated with the defect</li> </ul>	Defect owner, defect originator, duplicate defect originators, subscribers
DefectClose	Automatic action; no authority is required	Defect owner, defect originator, duplicate defect originators, subscribers
Not applicable; this is a base authority that can be performed by all users in the family	Defect owner, defect originator, duplicate defect originators, subscribers	
DefectDesign	<ul style="list-style-type: none"> <li>• Defect owner</li> <li>• Explicitly defined for the component associated with the defect</li> </ul>	Defect owner, defect originator, duplicate defect originators, subscribers
DefectModify	<ul style="list-style-type: none"> <li>• Defect owner can modify: <ul style="list-style-type: none"> <li>– answer, abstract, environment, driver, prefix, reference, release, and all configurable fields</li> </ul> </li> </ul> <p>Defect originator can modify:</p> <ul style="list-style-type: none"> <li>– originator, severity, name, abstract, environment, driver, prefix, reference, release, and all configurable fields</li> </ul> <ul style="list-style-type: none"> <li>• Explicitly defined for the component associated with the defect, these users can modify: <ul style="list-style-type: none"> <li>– abstract, answer, name, environment, driver, originator, prefix, reference, release, severity, phaseFound*, phaseInject*, priority*, symptom*, and target*</li> </ul> </li> </ul> <p>*If these fields have been configured by the family administrator, the field names might differ from those shown.</p>	Defect owner, defect originator, duplicate defect originators, subscribers

For this action	These users have authority	These users are notified
DefectOpen	Not applicable; this is a base authority that can be performed by all users in the family	Component owner, subscribers
DefectReopen	<ul style="list-style-type: none"> <li>Defect originator</li> <li>Explicitly defined for the component associated with the defect</li> </ul>	Defect owner, defect originator, duplicate defect originators, subscribers
DefectReturn	<ul style="list-style-type: none"> <li>Defect owner</li> <li>Explicitly defined for the component associated with the defect</li> </ul>	Defect originator, duplicate defect originators, subscribers
DefectReview	<ul style="list-style-type: none"> <li>Defect owner</li> <li>Explicitly defined for the component associated with the defect</li> </ul>	Defect owner, defect originator, duplicate defect originators, subscribers
DefectSize	<ul style="list-style-type: none"> <li>Defect owner</li> <li>Explicitly defined for the component associated with the defect</li> </ul>	Defect owner, defect originator, duplicate defect originators, subscribers
DefectVerify	<ul style="list-style-type: none"> <li>Defect owner</li> <li>Explicitly defined for the component associated with the defect</li> </ul>	Defect owner, defect originator, duplicate defect originators, subscribers
DefectView	<ul style="list-style-type: none"> <li>Defect owner, defect originator</li> <li>Explicitly defined for the component associated with the defect</li> </ul>	Not applicable
DriverAssign	<ul style="list-style-type: none"> <li>Driver owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	New owner, subscribers
DriverCheck	<ul style="list-style-type: none"> <li>Driver owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Not applicable
DriverCommit	<ul style="list-style-type: none"> <li>Explicitly defined for the component associated with the release</li> </ul>	Subscribers
DriverComplete	<ul style="list-style-type: none"> <li>Explicitly defined for the component associated with the release</li> </ul>	Subscribers

For this action	These users have authority	These users are notified
DriverCreate	<ul style="list-style-type: none"> <li>Release owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Subscribers
DriverDelete	<ul style="list-style-type: none"> <li>Driver owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Subscribers
DriverExtract	<ul style="list-style-type: none"> <li>Driver owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Not applicable
DriverFreeze	<ul style="list-style-type: none"> <li>Driver owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Driver owner, subscribers
DriverModify	<ul style="list-style-type: none"> <li>Driver owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Driver owner, subscribers
DriverRefresh	<ul style="list-style-type: none"> <li>Explicitly defined for the component associated with the release</li> </ul>	Component owner, subscribers
DriverRestrict	<ul style="list-style-type: none"> <li>Driver owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Driver owner, subscribers
DriverView	<ul style="list-style-type: none"> <li>Driver owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Not applicable
EnvCreate	<ul style="list-style-type: none"> <li>Release owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Tester, subscribers
EnvDelete	<ul style="list-style-type: none"> <li>Release owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Subscribers

For this action	These users have authority	These users are notified
EnvModify	<ul style="list-style-type: none"> <li>Release owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Tester, subscribers
FeatureAccept	<ul style="list-style-type: none"> <li>Feature owner</li> <li>Explicitly defined for the component associated with the feature</li> </ul>	Feature owner, feature originator, duplicate feature originators, subscribers
FeatureAssign	<ul style="list-style-type: none"> <li>Feature owner</li> <li>Explicitly defined for the component associated with the feature</li> </ul>	New owner, feature originator, duplicate feature originators, subscribers
FeatureCancel	<ul style="list-style-type: none"> <li>Feature originator</li> <li>Explicitly defined for the component associated with the feature</li> </ul>	Feature owner, feature originator, duplicate feature originators, subscribers
FeatureClose	Occurs automatically; no authority is required	Feature owner, feature originator, duplicate feature originators, subscribers
FeatureComment	Not applicable; this is a base authority that can be performed by all users in the family	Feature owner, feature originator, duplicate feature originators, subscribers
FeatureDesign	<ul style="list-style-type: none"> <li>Feature owner</li> <li>Explicitly defined for the component associated with the feature</li> </ul>	Feature owner, feature originator, duplicate feature originators, subscribers
FeatureModify	<ul style="list-style-type: none"> <li>Feature owner can modify: <ul style="list-style-type: none"> <li>abstract, prefix, reference, and all configurable fields</li> </ul> </li> <li>Feature originator can modify: <ul style="list-style-type: none"> <li>abstract, name, prefix, reference, and all configurable fields</li> </ul> </li> <li>Explicitly defined for the component associated with the feature, these users can modify: <ul style="list-style-type: none"> <li>abstract, name, originator, prefix, reference, priority*, and target*</li> </ul> </li> </ul> <p>*If these fields have been configured by the family administrator, the field names might differ from those shown.</p>	Feature owner, feature originator, duplicate feature originators, subscribers
FeatureOpen	Not applicable; this is a base authority that can be performed by all users in the family	Component owner, subscribers

For this action	These users have authority	These users are notified
FeatureReopen	<ul style="list-style-type: none"> <li>Feature originator</li> <li>Explicitly defined for the component associated with the feature</li> </ul>	Feature owner, feature originator, duplicate feature originators, subscribers
FeatureReturn	<ul style="list-style-type: none"> <li>Feature owner</li> <li>Explicitly defined for the component associated with the feature</li> </ul>	Feature owner, feature originator, duplicate feature originators, subscribers
FeatureReview	<ul style="list-style-type: none"> <li>Feature owner</li> <li>Explicitly defined for the component associated with the feature</li> </ul>	Feature owner, feature originator, duplicate feature originators, subscribers
FeatureSize	<ul style="list-style-type: none"> <li>Feature owner</li> <li>Explicitly defined for the component associated with the feature</li> </ul>	Feature owner, feature originator, duplicate feature originators, subscribers
FeatureVerify	<ul style="list-style-type: none"> <li>Feature owner</li> <li>Explicitly defined for the component associated with the feature</li> </ul>	Feature owner, feature originator, duplicate feature originators, subscribers
FeatureView	<ul style="list-style-type: none"> <li>Feature owner</li> <li>Explicitly defined for the component associated with the feature</li> </ul>	Not applicable
FixActive	<ul style="list-style-type: none"> <li>Fix record owner, component owner, work area owner</li> <li>Explicitly defined for the component associated with the fix record</li> </ul>	Subscribers
FixAssign	<ul style="list-style-type: none"> <li>Fix record owner, component owner, work area owner</li> <li>Explicitly defined for the component associated with the fix record</li> </ul>	New fix record owner, subscribers
FixComplete	<ul style="list-style-type: none"> <li>Fix record owner, component owner, work area owner</li> <li>Explicitly defined for the component associated with the fix record</li> </ul>	Subscribers
FixCreate	<ul style="list-style-type: none"> <li>Defect or feature owner, work area owner</li> <li>Explicitly defined for the component associated with the defect or feature</li> </ul>	Subscribers



For this action	These users have authority	These users are notified
FixDelete	<ul style="list-style-type: none"> <li>Defect or feature owner, work area owner</li> <li>Explicitly defined for the component associated with the defect or feature</li> </ul>	Subscribers
HostCreate	<ul style="list-style-type: none"> <li>Owner of the user ID for which a host list entry is being created or deleted</li> <li>Superuser</li> </ul>	Not applicable
HostDelete	<ul style="list-style-type: none"> <li>Owner of the user ID for which a host list entry is being deleted</li> <li>Superuser</li> </ul>	Not applicable
MemberCreate	<ul style="list-style-type: none"> <li>Driver owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Driver owner, subscribers
MemberCreateR	<ul style="list-style-type: none"> <li>Driver owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Driver owner, subscribers
MemberDelete	<ul style="list-style-type: none"> <li>Driver owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Driver owner, subscribers
MemberDeleteR	<ul style="list-style-type: none"> <li>Driver owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Driver owner, subscribers
NotifyCreate	<ul style="list-style-type: none"> <li>Component owner</li> <li>Explicitly defined for the component associated with the notification list</li> </ul>	Not applicable
NotifyDelete	<ul style="list-style-type: none"> <li>Component owner</li> <li>Owner of user ID</li> <li>Explicitly defined for the component associated with the notification list</li> </ul> <p><b>Note:</b> Users can delete themselves from a notification list without requiring any authority</p>	Not applicable
ParserCreate	<ul style="list-style-type: none"> <li>Explicitly defined for the component associated with the release</li> </ul>	Subscribers

For this action	These users have authority	These users are notified
ParserDelete	<ul style="list-style-type: none"> <li>Explicitly defined for the component associated with the release</li> </ul>	Subscribers
ParserModify	<ul style="list-style-type: none"> <li>Explicitly defined for the component associated with the release</li> </ul>	Subscribers
ParserView	<ul style="list-style-type: none"> <li>Explicitly defined for the component associated with the release</li> </ul>	Not applicable
PartAdd	<ul style="list-style-type: none"> <li>Component owner</li> <li>Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartBuild	<ul style="list-style-type: none"> <li>Component owner</li> <li>Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartCheckIn	<ul style="list-style-type: none"> <li>User who checked out or locked the part originally, component owner</li> <li>Explicitly defined for the component associated with the part</li> </ul> <p><b>Note:</b> The user who is explicitly given this authority can check in a part that is checked out by someone else.</p>	Subscribers
PartCheckOut	<ul style="list-style-type: none"> <li>Component owner</li> <li>Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartChildInfo	<ul style="list-style-type: none"> <li>Component owner</li> <li>Explicitly defined for the component associated with the part</li> </ul>	Not applicable
PartConnect	<ul style="list-style-type: none"> <li>Component owner</li> <li>Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartDelete	<ul style="list-style-type: none"> <li>Component owner</li> <li>Explicitly defined for the component associated with the part</li> </ul>	Subscribers

For this action	These users have authority	These users are notified
PartDeleteForce	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartExtract	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Not applicable
PartForceIn	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartForceOut	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartLink	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartLock	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartLockForce	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartModify	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartRecreateForce	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartRecreate	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Subscribers

For this action	These users have authority	These users are notified
PartRefresh	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartRename	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartRenameForce	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartResolve	Not applicable; this is a base authority that can be performed by all users in the family	Not applicable
PartTouch	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartUndo	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartUndoForce	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartUnlock	<ul style="list-style-type: none"> <li>• User who checked out or locked the part originally, component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Subscribers
PartView	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Not applicable
PartViewMsg	<ul style="list-style-type: none"> <li>• Component owner</li> <li>• Explicitly defined for the component associated with the part</li> </ul>	Not applicable
PrereqCreate	<ul style="list-style-type: none"> <li>• Work area owner of all specified work areas</li> <li>• Explicitly defined for the component managing the associated work area and release</li> </ul>	Not applicable

For this action	These users have authority	These users are notified
PrereqDelete	<ul style="list-style-type: none"> <li>Work area owner of all specified work areas</li> <li>Explicitly defined for the component managing the associated work area and release</li> </ul>	Not applicable
ReleaseCreate	<ul style="list-style-type: none"> <li>Explicitly defined for the component associated with the new release</li> </ul>	New release owner, component owner, subscribers
ReleaseDelete	<ul style="list-style-type: none"> <li>Release owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Release owner, component owner, subscribers
ReleaseExtract	<ul style="list-style-type: none"> <li>Release owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Not applicable
ReleaseLink	<ul style="list-style-type: none"> <li>Release owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Release owner, subscribers
ReleaseModify	<ul style="list-style-type: none"> <li>Release owner</li> <li>Explicitly defined for the component associated with the release</li> </ul> <p><b>Note:</b> To identify a new component to manage the release, you must have ReleaseCreate in an authority group in the component that you are modifying</p>	Release owner, subscribers, new owner (if applicable)
ReleasePrune	<ul style="list-style-type: none"> <li>Release owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Subscribers
ReleaseRecreate	<ul style="list-style-type: none"> <li>Release owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Release owner, component owner, subscribers
ReleaseView	<ul style="list-style-type: none"> <li>Release owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Not applicable
Report	Not applicable; this is a base authority that can be performed by all users in the family	Not applicable

For this action	These users have authority	These users are notified
SizeAccept	<ul style="list-style-type: none"> <li>Sizing record owner</li> <li>Explicitly defined for the component associated with the sizing record</li> </ul>	Subscribers
SizeAssign	<ul style="list-style-type: none"> <li>Sizing record owner</li> <li>Explicitly defined for the component associated with the sizing record</li> </ul>	New sizing record owner, defect/feature owner, subscribers
SizeCreate	<ul style="list-style-type: none"> <li>Defect/feature owner</li> <li>Explicitly defined for the component associated with the defect/feature</li> </ul>	Component owner, defect/feature owner, subscribers
SizeDelete	<ul style="list-style-type: none"> <li>Defect/feature owner</li> <li>Explicitly defined for the component associated with the defect/feature</li> </ul>	Subscribers, sizing record owner, defect/feature owner
SizeReject	<ul style="list-style-type: none"> <li>Sizing record owner</li> <li>Explicitly defined for the component associated with the sizing record</li> </ul>	Subscribers
TestAbstain	<ul style="list-style-type: none"> <li>Test record owner</li> <li>Explicitly defined for the component associated with the test record's release</li> </ul>	Subscribers
TestAccept	<ul style="list-style-type: none"> <li>Test record owner</li> <li>Explicitly defined for the component associated with the test record's release</li> </ul>	Subscribers
TestAssign	<ul style="list-style-type: none"> <li>Test record owner</li> <li>Explicitly defined for the component associated with the test record's release</li> </ul>	New test record owner, subscribers
TestReady	<ul style="list-style-type: none"> <li>Test record owner</li> <li>Explicitly defined for the component associated with the test record's release</li> </ul>	Subscribers
TestReject	<ul style="list-style-type: none"> <li>Test record owner</li> <li>Explicitly defined for the component associated with the test record's release</li> </ul>	Subscribers
UserCreate	Superuser	New user
UserDelete	Superuser	Not applicable

For this action	These users have authority	These users are notified
UserModify	<ul style="list-style-type: none"> <li>Owner of the user object can modify all characteristics except the superuser privilege</li> <li>Must be a superuser to grant the superuser privilege</li> </ul>	Not applicable
UserRecreate	Superuser	Not applicable
UserView	Not applicable; this is a base authority that can be performed by all users in the family	Not applicable
VerifyAbstain	<ul style="list-style-type: none"> <li>Verification record owner</li> <li>Explicitly defined for the component associated with the verification record's defect or feature</li> </ul>	Subscribers
VerifyAccept	<ul style="list-style-type: none"> <li>Verification record owner</li> <li>Explicitly defined for the component associated with the verification record's defect or feature</li> </ul>	Subscribers
VerifyAssign	<ul style="list-style-type: none"> <li>Verification record owner</li> <li>Explicitly defined for the component associated with the verification record's defect or feature</li> </ul>	New verification record owner, subscribers
VerifyReady	Takes place automatically; no authority is required	Verification record owners
VerifyReject	<ul style="list-style-type: none"> <li>Verification record owner</li> <li>Explicitly defined for the component associated with the verification record's defect or feature</li> </ul>	Subscribers
WorkAreaAssign	<ul style="list-style-type: none"> <li>Work area owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	New work area owner, subscribers
WorkAreaCancel	<ul style="list-style-type: none"> <li>Defect or feature owner</li> <li>Explicitly defined for the component associated with the defect or feature</li> </ul>	Subscribers, owners of approval records for work area being canceled
WorkAreaCheck	<ul style="list-style-type: none"> <li>Work area owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Not applicable
WorkAreaCommit	<ul style="list-style-type: none"> <li>Work area owner</li> <li>Explicitly defined for the component associated with the release</li> </ul>	Subscribers

For this action	These users have authority	These users are notified
WorkAreaComple	<ul style="list-style-type: none"> <li>• Work area owner</li> <li>• Explicitly defined for the component associated with the release</li> </ul>	Subscribers
WorkAreaCreate	<ul style="list-style-type: none"> <li>• Defect or feature owner</li> <li>• Explicitly defined for the component associated with the defect or feature</li> </ul>	Work area owner, subscribers
WorkAreaFix	<ul style="list-style-type: none"> <li>• Work area owner</li> <li>• Explicitly defined for the component associated with the release</li> </ul>	Subscribers
WorkAreaFreeze	<ul style="list-style-type: none"> <li>• Work area owner</li> <li>• Explicitly defined for the component associated with the release</li> </ul>	Subscribers
WorkAreaIntegra	<ul style="list-style-type: none"> <li>• Work area owner</li> <li>• Explicitly defined for the component associated with the release</li> </ul>	Subscribers
WorkAreaModify	<ul style="list-style-type: none"> <li>• Work area owner</li> <li>• Explicitly defined for the component associated with the release</li> </ul>	Subscribers
WorkAreaRefresh	<ul style="list-style-type: none"> <li>• Work area owner</li> <li>• Explicitly defined for the component associated with the release</li> </ul>	Work area owner, subscribers
WorkAreaTest	<ul style="list-style-type: none"> <li>• Work area owner</li> <li>• Explicitly defined for the component associated with the release</li> </ul>	Subscribers
WorkAreaUndo	<ul style="list-style-type: none"> <li>• Work area owner</li> <li>• Explicitly defined for the component associated with the release</li> </ul>	
WorkAreaView	<ul style="list-style-type: none"> <li>• Work area owner</li> <li>• Explicitly defined for the component associated with the release</li> </ul>	Not applicable



---

## Appendix G. Sample REXX execs, build scripts, and parsers

This appendix is composed of the IBM-supplied REXX execs, build scripts, and parsers. Your family administrator can modify these samples to fit the needs of your organization.

The samples in this appendix may not be available on all platforms. Refer to the readme file for a complete list of samples available with TeamConnection. All samples are provided as-is and any use of or modifications to the samples are the sole responsibility of the customer.

---

### Sample REXX execs

This section lists the sample REXX execs that are shipped with TeamConnection. The client.smp file contains this same listing. It is located in the bin subdirectory of the directory where the TeamConnection client is installed.

Users running these execs must have user and host access to your TeamConnection family.

Most of the execs require input parameters, and some require that the TC\_FAMILY or TC\_RELEASE environment variables be set. If the user who is running the script is acting for another user, the TC\_BECOME environment variable must also be set. These variables can be set from a command line prompt.

The following convention is used to show the required, optional, and selective input parameters:

- Brackets ( [ ] ) indicate that the input or variable is optional.
- Braces ( { } ) indicate that one of the inputs is required.
- An input or variable that is not surrounded by brackets or braces is required.

Script name	Function	Inputs	Environment variables
accComp	Lists the explicit access of users in a specified component.	componentName	TC_FAMILY [TC_BECOME]
compChld	Lists the direct children of a specified component. Also lists the description and owner of each child component.	componentName	TC_FAMILY [TC_BECOME]
compOwnr	Displays a list of component owners' addresses in a specified family.	familyName	[TC_BECOME]
compPrnt	Lists the parent component of a specified component.	componentName	TC_FAMILY [TC_BECOME]
compWalk	Displays the children and grandchildren of a specified component.	componentName	TC_FAMILY [TC_BECOME]
defClone	Creates a new defect based on values contained in a specified defect.	defectNumber	TC_FAMILY [TC_BECOME]
defDvr	Lists all defects for a specified driver.	driverName	TC_FAMILY [TC_BECOME]
defFfea	Creates a new defect based on values contained in a specified feature.	featureNumber	TC_FAMILY [TC_BECOME]
defNew	Displays the number of the most recent defect that was entered in the system.		TC_FAMILY [TC_BECOME]

Script name	Function	Inputs	Environment variables
defReopn	Reopens a previously canceled or returned defect.	defectNumber	TC_FAMILY [TC_BECOME]
defRept	Generates a global defect report showing work areas, test records, approval records, and fix records.		TC_FAMILY [TC_BECOME]
defState	Lists the defect number of all defects that are in a specified state.	stateName	TC_FAMILY [TC_BECOME]
defStats	Displays total active defect statistics on a defect owner area basis.	ownerArea	TC_FAMILY [TC_BECOME]
defWRef	Displays the full details of defects that contain the specified reference field value.	reference	TC_FAMILY [TC_BECOME]
dfDesc	Displays the full remarks that were entered when the specified defect or feature was created.	{defectNumber   featureNumber}	TC_FAMILY [TC_BECOME]
feaClone	Creates a new feature based on values contained in a specified feature.	featureNumber	TC_FAMILY [TC_BECOME]
feaDvr	Lists all features contained in a specified driver.	driverName	TC_FAMILY [TC_BECOME]
feaFdef	Creates a new feature based on the values contained in the specified defect.	defectNumber	TC_FAMILY [TC_BECOME]
feaNew	Displays the number of the most recent feature that was entered in the system.		TC_FAMILY [TC_BECOME]
feaReopn	Reopens a previously canceled or returned feature.	featureNumber	TC_FAMILY [TC_BECOME]
feaRept	Generates a global feature report showing work areas, test records, size records, and fix records.		TC_FAMILY [TC_BECOME]
feaState	Lists the feature number of all features that are in a specified state.	stateName	TC_FAMILY [TC_BECOME]
feaStats	Displays total active feature statistics on a feature owner area basis.	ownerArea	TC_FAMILY [TC_BECOME]
drvByDF	Lists the name of the drivers that contain a specified defect or feature.	{defectNumber   featureNumber}	TC_FAMILY [TC_BECOME]
drvMem	Lists the defect and feature members of a specified driver for a specified release.	driverName [releaseName]	TC_FAMILY [TC_RELEASE] [TC_BECOME]
mailTo	Sends a message to the addresses read through stdin.	messagefile subject	
ownerChg	Re-assigns all current work and objects owned by userLogin1 to userLogin2.	userLogin1 userLogin2	TC_FAMILY [TC_BECOME]
prtChckin	Checks parts into the TeamConnection family. When common parts are encountered, the script requests the releases for which the part should remain common.	partPathName [releaseName]	TC_FAMILY [TC_RELEASE] [TC_BECOME]
prtChgDf	Lists all the parts that were changed for a specified defect or feature.	{defectNumber   featureNumber}	TC_FAMILY [TC_BECOME]
prtChgDr	Lists the parts that were changed for a specified driver.	driverName	TC_FAMILY [TC_BECOME]

Script name	Function	Inputs	Environment variables
prtComGt	Extracts all the parts associated with a specific component. The parts are placed in a directory that represents the release name to which the version of the part is associated. This directory is created relative to the relativePathName parameter.	componentName relativePathName [committed]	TC_FAMILY [TC_BECOME]
prtComp	Lists all parts related to a specified component.	componentName	TC_FAMILY [TC_BECOME]
prtHist	Lists all defect and feature numbers and abstracts that caused a change to a specified part in a specified release.	partName [releaseName]	TC_FAMILY [TC_RELEASE] [TC_BECOME]
prtInfo	Displays information for a specified part.	partName	TC_FAMILY [TC_RELEASE] [TC_BECOME]
prtLock	Lists all parts that a specified user has locked.	userLogin	TC_FAMILY [TC_BECOME]
prtLokBy	Lists who has a specified part checked out.	partName	TC_FAMILY TC_RELEASE [TC_BECOME]
PrtPath	Finds and lists all parts that match a partial part path name.	partPathName	TC_FAMILY [TC_BECOME]
prtRel	Lists all parts related to a specified release.	releaseName	TC_FAMILY [TC_BECOME]
prtWaGt	Extracts all the parts associated with a specific work area and places them in the path specified by the relativePathName parameter.	releaseName workareaName relativePathName	TC_FAMILY [TC_BECOME]
rByArea	Generates a manager's report based on the specified areas or departments of interest.	areaName ...	TC_FAMILY [TC_BECOME]
relOwner	Displays a list of addresses of all release owners in a specified TeamConnection family.	familyName	[TC_BECOME]
showConf	Lists the valid values pertaining to a specified configurable type.	configType	TC_FAMILY [TC_BECOME]
userAuth	Lists the users who have the authority to give other users access to a specified component.	componentName	TC_FAMILY [TC_BECOME]
userInfo	Finds user information based on part of the user's name. A fuzzy search is performed.	userName	TC_FAMILY [TC_BECOME]
usersAll	Lists the addresses of all users in a specified TeamConnection family.	familyName	[TC_BECOME]
usrAcc	Lists the explicit access of a specified user for the specified component and its descendant components.	userLogin componentName	TC_FAMILY [TC_BECOME]
usrRept	Generates a user's report based on the specified user login.	userLogin	TC_FAMILY [TC_BECOME]
verByPrt	Lists the version numbers, release names, and path names for the specified part.	partName releaseName	TC_FAMILY [TC_BECOME]
waComit	Lists the work areas that are in the commit state for a specified release.	releaseName	TC_FAMILY [TC_BECOME]
waFix	Lists all the work areas that are in the fix state for a given release.	releaseName	TC_FAMILY [TC_BECOME]

Script name	Function	Inputs	Environment variables
waInLvl	Lists the work areas that are in the integrate state and are associated with at least one development driver for the specified release.	releaseName	TC_FAMILY [TC_BECOME]
waInt	Lists the work areas that are in the integrate state for a specified release.	releaseName	TC_FAMILY [TC_BECOME]
waPrdLv	Lists the work areas that are included in a production driver and are in the integrate state for a specified release.	releaseName	TC_FAMILY [TC_BECOME]
waStat	Generates a work area activity statistics report on a user area basis.	userArea	TC_FAMILY [TC_BECOME]
waTest	Lists the work areas that are in the test state for a specified release.	releaseName	TC_FAMILY [TC_BECOME]

## Sample build scripts

### **fhbcob2.cmd**

Calls the COBOL Visual Set for OS/2 compiler.

### **fhbcob2l.cmd**

Calls the COBOL Visual Set for OS/2 compiler and link editor.

### **fhbocomp.cmd**

Calls the VisualAge for C++ icc compiler.

### **fhbolib.cmd**

Calls the OS/2 implib utility.

### **fhbolin2.cmd**

Calls the VisualAge for C++ icc link editor.

### **fhbolink.cmd**

Calls the link386 link editor.

### **fhborc.cmd**

Calls the OS/2 resource compiler.

### **fhbplbld.cmd**

Calls the OS/2 PL/1 compiler.

### **fhbpllnk.cmd**

Calls the OS/2 PL/1 link editor.

### **edcc.jcl**

Calls the C/370 JCL procedure.

### **fhbcobm.jcl**

Calls the COBOL for MVS compiler.

### **fhbm370.jcl**

Calls the C/370 compiler.

### **fhbmasm.jcl**

Calls the MVS assembler.

### **fhbmc.jcl**

Calls the C/370 compiler.

### **fhbmlink**

Calls the MVS linkage editor.

**fhbmpli.jcl**  
Calls the PL/1 MVS compiler.

**fhbplked.jcl**  
Calls the C370 prelinker.

**fhbtclnk**  
Calls the TeamConnection pseudo linker.

**fhbwcomp.c**  
Calls the Microsoft Visual C++ compiler

**fhbwlink.c**  
Calls the Microsoft linker

**gather.cmd**  
Calls the Gather tool.

**nvbridge.cmd**  
Calls the NetView bridge tool (NVBridge).

---

## Sample parsers

**fhbcbprs.cmd**  
A parser for COBOL applications.

**fhbopars.cmd**  
A parser for C applications.

**fhbplprs.cmd**  
A parser for PL/1 applications.

---

## Sample package files

**gather.pkf**  
A package file for the Gather tool.

**nvbridge.pkf**  
A package file for the NetView bridge tool (NVBridge).



## Appendix H. Worksheets

### Authority groups worksheet

The following table lists TeamConnection actions. Use this table to record the authority groups for your family if you are using groups other than those supplied by IBM. Those TeamConnection actions that cannot be included in an authority group are marked with information about how they can be performed.

TeamConnection action	Authority groups for family: _____									
AccessCreate										
AccessDelete										
AccessRestrict										
ApprovalAbstain										
ApprovalAccept										
ApprovalAssign										
ApprovalCreate										
ApprovalDelete										
ApprovalReject										
ApproverCreate										
ApproverDelete										
BuilderCreate										
BuilderDelete										
BuilderExtract										
BuilderModify										
BuilderView										
CollisionAccept										
CollisionReconc										
CollisionReject										
CompCreate										
CompDelete										
CompLink										
CompModify										
CompRecreate										
CompUnlink										
CompView										
CoreqCreate										
CoreqDelete										
DefectAccept										
DefectAssign										

TeamConnection action	Authority groups for family: _____									
DefectCancel										
DefectClose	Automatic action									
DefectComment	Base authority									
DefectDesign										
DefectModify										
DefectOpen	Base authority									
DefectReopen										
DefectReturn										
DefectReview										
DefectSize										
DefectVerify										
DefectView										
DriverAssign										
DriverCheck										
DriverCommit										
DriverComplete										
DriverCreate										
DriverDelete										
DriverExtract										
DriverFreeze										
DriverModify										
DriverRefresh										
DriverRestrict										
DriverView										
EnvCreate										
EnvDelete										
EnvModify										
FeatureAccept										
FeatureAssign										
FeatureCancel										
FeatureClose	Automatic action									
FeatureComment	Base authority									
FeatureDesign										
FeatureModify										
FeatureOpen	Base authority									
FeatureReopen										
FeatureReturn										



TeamConnection action	Authority groups for family: _____									
FeatureReview										
FeatureSize										
FeatureVerify										
FeatureView										
FixActive										
FixAssign										
FixComplete										
FixCreate										
FixDelete										
HostCreate	Superuser or owner implicit authority									
HostDelete	Superuser or owner implicit authority									
MemberCreate										
MemberCreateR										
MemberDelete										
MemberDeleteR										
NotifyCreate										
NotifyDelete										
ParserCreate										
ParserDelete										
ParserModify										
ParserView										
PartAdd										
PartBuild										
PartCheckIn										
PartCheckOut										
PartChildInfo										
PartConnect										
PartDelete										
PartDeleteForce										
PartDisconnect										
PartExtract										
PartForceIn										
PartForceOut										
PartLink										
PartLock										
PartLockForce										
PartModify										

TeamConnection action	Authority groups for family: _____									
PartRecreate										
PartRecreaForce										
PartRename										
PartRenameForce										
PartResolve	Base authority									
PartTouch										
PartUndo										
PartUndoForce										
PartUnlock										
PartView										
PartViewmsg										
PrereqCreate										
PrereqDelete										
ReleaseCreate										
ReleaseDelete										
ReleaseExtract										
ReleaseLink										
ReleaseModify										
ReleasePrune										
ReleaseRecreate										
ReleaseView										
Report	Base authority									
SizeAccept										
SizeAssign										
SizeCreate										
SizeDelete										
SizeReject										
TestAbstain										
TestAccept										
TestAssign										
TestReady	Automatic action									
TestReject										
UserCreate	Superuser implicit authority									
UserDelete	Superuser implicit authority									
UserModify	Superuser or owner implicit authority									
UserRecreate	Superuser implicit authority									
UserView										

TeamConnection action	Authority groups for family: _____									
VerifyAbstain										
VerifyAccept										
VerifyAssign										
VerifyReady	Automatic action									
VerifyReject										
WorkAreaAssign										
WorkAreaCancel										
WorkAreaCheck										
WorkAreaCommit										
WorkAreaComple										
WorkAreaCreate										
WorkAreaFix										
WorkAreaFreeze										
WorkAreaIntegra										
WorkAreaModify										
WorkAreaRefresh										
WorkAreaTest										
WorkAreaView										

## Interest groups worksheet

The following table lists the TeamConnection actions. Use this table to record the interest groups for your family if you are using groups other than those supplied by IBM. Those TeamConnection actions that cannot be included in an interest group are marked with information about how users are notified.

TeamConnection actions	Interest groups for family: _____						
AccessCreate							.
AccessDelete							
AccessRestrict							.
ApprovalAbstain							
ApprovalAccept							
ApprovalAssign							
ApprovalCreate							
ApprovalDelete							
ApprovalReject							
ApproverCreate							
ApproverDelete							

TeamConnection actions	Interest groups for family: _____						
BuilderCreate							
BuilderDelete							
BuilderExtract	No notification						
BuilderModify							
BuilderView	No notification						
CollisionAccept							
CollisionReconc							
CollisionReject							
CompCreate	New owner implicit notification						
CompDelete							
CompLink							
CompModify							
CompRecreate							
CompUnlink							
CompView	No notification						
CoreqCreate	No notification						
CoreqDelete	No notification						
DefectAccept							
DefectAssign							
DefectCancel							
DefectClose							
DefectComment							
DefectDesign							
DefectModify							
DefectOpen							
DefectReopen							
DefectReturn							
DefectReview							
DefectSize							
DefectVerify							
DefectView	No notification						
DriverAssign							
DriverCheck	No notification						
DriverCommit							
DriverComplete							
DriverCreate							
DriverDelete							

TeamConnection actions	Interest groups for family: _____						
DriverExtract	No notification						
DriverFreeze							
DriverModify							
DriverRefresh	No notification						
DriverRestrict	Owner implicit notification						
DriverView	No notification						
EnvCreate							
EnvDelete							
EnvModify							
FeatureAccept							
FeatureAssign							
FeatureCancel							
FeatureClose							
FeatureComment							
FeatureDesign							
FeatureModify							
FeatureOpen							
FeatureReopen							
FeatureReturn							
FeatureReview							
FeatureSize							
FeatureVerify							
FeatureView	No notification						
FixActive							
FixAssign							
FixComplete							
FixCreate							
FixDelete							
HostCreate	No notification						
HostDelete	No notification						
MemberCreate							
MemberCreateR	New owner implicit notification						
MemberDelete							
MemberDeleteR	Owner implicit notification						
NotifyCreate	No notification						
NotifyDelete	No notification						
ParserCreate							

TeamConnection actions	Interest groups for family: _____						
ParserDelete							
ParserModify							
ParserView	No notification						
PartAdd							
PartBuild	owner implicit notification						
PartCheckIn							
PartCheckOut							
PartConnect							
PartDelete							
PartDisconnect							
PartExtract	No notification						
PartForceIn							
PartForceOut							
PartLink							
PartLock							
PartLockForce	PartLock subscribers						
PartModify							
PartRecreaForce	PartRecreate subscribers						
PartRecreate							
PartRename							
PartRenameForce	PartRename subscribers						
PartResolve	No notification						
PartTouch	No notification						
PartUndo							
PartUndoForce	PartUndo subscribers						
PartUnlock							
PartView	No notification						
PartViewmsg	No notification						
PrereqCreate	New owner implicit notification						
PrereqDelete	Owner implicit notification						
ReleaseCreate							
ReleaseDelete							
ReleaseExtract	No notification						
ReleaseLink							
ReleaseModify							
ReleaseRecreate							
ReleaseView	No notification						

TeamConnection actions	Interest groups for family: _____						
Report	No notification						
SizeAccept							
SizeAssign							
SizeCreate							
SizeDelete							
SizeReject							
TestAbstain							
TestAccept							
TestAssign							
TestReady	Owner implicit notification						
TestReject							
UserCreate	New user implicit notification						
UserDelete	No notification						
UserModify	No notification						
UserUnDelete	No notification						
UserView	No notification						
VerifyAbstain							
VerifyAccept							
VerifyAssign							
VerifyReady	Owner implicit notification						
VerifyReject							
WorkAreaAssign							
WorkAreaCancel							
WorkAreaCheck	No notification						
WorkAreaCommit							
WorkAreaCompleat							
WorkAreaCreate							
WorkAreaFix							
WorkAreaFreeze	Owner implicit notification						
WorkAreaIntegra							
WorkAreaModify							
WorkAreaRefresh							
WorkAreaTest							
WorkAreaView	No notification						

---

## Configurable processes worksheets

The following worksheets list the TeamConnection subprocesses. Use these worksheets to record the processes that you have created for your family. Separate worksheets are provided for component and release processes. For more information on configuring processes, see “Chapter 14. Configuring family processes” on page 153.

TeamConnection component subprocesses	Component processes for _____					
dsrDefect						
dsrFeature						
verifyDefect						
verifyFeature						
none						

TeamConnection release subprocesses	Release processes for _____					
track						
approval						
fix						
driver						
test						
none						



---

## Customer support

Your options for IBM VisualAge TeamConnection support, as described in your License Information and Licensed Program Specifications, include electronic forums. You can use the electronic forums to access IBM VisualAge TeamConnection technical information, exchange messages with other TeamConnection users, and receive information regarding the availability of fixes. The following forums are available.

- **IBM Talklink**

Use the TEAMC CFORUM. For additional information about TalkLink, call

- United States 1-800-547-1283
- Canada 1-800-465-7999 ext. 228

- **CompuServe**

From any ! prompt, type GO SOFSOL, then select TeamConnection. For additional information, call 1-800-848-8199 and ask for representative 239.

- **Internet**

Go to the IBM homepage, <http://www.ibm.com>. Use the search function with keyword TeamConnection to go to the TeamConnection area.

If you cannot access these forums, contact your IBM representative.

There are several other support offerings available after purchasing IBM VisualAge TeamConnection. For a list of these offerings, please contact your IBM representative.



---

## Bibliography

---

### IBM VisualAge TeamConnection library

The following is a list of the TeamConnection publications.

- **License Information (GC34-4497):**  
Contains license, service, and warranty information.
- **Administrator's Guide (GC34-4551):**  
Lists the hardware and software that are required before you can install and use the IBM VisualAge TeamConnection product, provides detailed instructions for installing and configuring the TeamConnection family and build servers, and provides instructions for administering a TeamConnection family.
- **Getting Started with the TeamConnection Clients (SC34-4552):**  
Tells first-time users how to install the TeamConnection clients on their workstations, and familiarizes them with the command line and graphical user interfaces.
- **User's Guide (SC34-4499):**  
A comprehensive guide for TeamConnection administrators and client users that helps them install and use TeamConnection.
- **Commands Reference (SC34-4501):**  
Describes the TeamConnection commands, their syntax, and the authority required to issue each command. This book also provides examples of how to use the various commands.
- **Quick Commands Reference (GC34-4500):**  
Lists the TeamConnection commands along with their syntax.
- **Staying on Track with TeamConnection Processes (83H9677):**  
Poster showing how objects flow through the states defined for each TeamConnection process.
- The following publications can be ordered as a set (SBOF-8560):
  - Administrator's Guide**
  - Getting Started with the TeamConnection Clients**
  - User's Guide**
  - Commands Reference**
  - Quick Commands Reference**
  - Staying on Track with TeamConnection Processes**

---

### Tool Builder's Development Kit

The following publications are part of the Tool Builder's Development Kit feature:

- **Tool Builder's Development Guide (SC34-4553):**  
Explains how to create and extend tools for accessing objects in the TeamConnection database. It contains guidance and reference information.
- **Information Model Reference (SC34-4554):**  
Details the TeamConnection information model. This publication is available in softcopy only.

---

## TeamConnection Technical reports

- 29.2147**  
SCLM Guide to TeamConnection Terminology
- 29.2196**  
Using REXX command files with TeamConnection MVS Build Scripts
- 29.2231**  
TeamConnection Interoperability with MVS and SCLM
- 29.2235**  
Using REXX command files with TeamConnection MVS Build Scripts for PL/I programs
- 29.2253**  
Comparison between CMVC 2.3 and TeamConnection 2
- 29.2254**  
Migrating from CMVC 2.3 to TeamConnection 2
- 29.2267**  
TeamConnection frequently asked questions: how to do routine operating system tasks

---

## ObjectStore

The following publications are part of the ObjectStore library of documents and are available for order from Object Design, Inc. To order these documents call (617) 674-5000, Monday through Friday from 8:30 AM to 5:30 PM Eastern Time.

- **ObjectStore C++ Installation:**  
Contains step-by-step procedures for installing the latest release of ObjectStore on a specific platform:
  - 310-100-40 I**  
UNIX
  - 310-310-40 I**  
Windows
  - 310-320-40 I**  
OS/2
- **ObjectStore C++ API User Guide (310-000-40 U):**  
Provides information about the application programming interface for application programmers.
- **ObjectStore C++ API Reference (310-000-40 R):**  
Describes the API to the features provided by ObjectStore for application programmers.
- **ObjectStore C++ Building Applications (310-000-40 B):**  
Provides information and instructions for compiling code, generating schemas, and linking files using all supported compilers; and provides instructions for developing ObjectStore client applications for use on multiple platforms.
- **ObjectStore Management (310-000-40 M):**  
Provides information and instructions for performing management tasks on ObjectStore server and client systems. It includes server parameters, environment variables, and database utilities.
- **ObjectStore C++ Performance (310-000-40 P):**

Explains the fundamentals of designing and tuning ObjectStore applications for optimal performance.

---

## IBM Exchange library

The publications listed below can be ordered as a set (SBOF-6098) or separately as indicated below. IBM Exchange will be available at a later date.

- *Licensed Programming Specification (GC34-4525):*
- *Installation Guide (SC34-4509):*
- *Bridge Builder's Guide (SC34-4508):*
- *User's Guide 1 (SC34-4506):*
- *User's Guide 2 (SC34-4507):*

---

## Related publications

- Transmission Control Protocol/Internet Protocol (TCP/IP)
  - *TCP/IP 2.0 for OS/2: Installation and Administration (SC31-6075)*
  - *TCP/IP for MVS Planning and Customization (SC31-6085)*
- MVS
  - *MVS/XA JCL User's Guide (GC28-1351)*
  - *MVS/XA JCL Reference (GC28-1352)*
  - *MVS/ESA JCL User's Guide (GC28-1830)*
  - *MVS/ESA JCL Reference (GC28-1829)*
- NLS and DBCS
  - *AIX 4, General Programming Concepts: Writing and Debugging Programs (SC23-2533-02)*. See chapter 16 "National Language Support" for an updated contents of the AIX 3 material (see below).
  - *AIX 4, System Management Guide: Operating System and Devices (SC23-2525-03)*. See chapter 10, "National Language Support" for system tasks.
  - *AIX Version 3.2 for RISC System/6000, National Language Support (GG24-3850)*.
  - *Internationalization of AIX Software, A Programmer's Guide (SC23-2431)*.
  - *National Language Design Guide Volume 1 (SE09-8001-02)*. This manual contains very good information on how to enable an application for NLS.
  - *National Language Design Guide Volume 2 (SE09-8002-02)*. This manual provides information on the IBM language codes (consult the "Language codes" chapter).



---

# Glossary

This glossary includes terms and definitions from the *IBM Dictionary of Computing*, 10th edition (New York: McGraw-Hill, 1993). If you do not find the term you are looking for, refer to this document's index or to the *IBM Dictionary of Computing*.

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

**absolute path name.** A directory or a part expressed as a sequence of directories followed by a part name beginning from the root directory.

**access list.** A set of objects that controls access to data. Each object consists of a component, a user, and the authority that the user is granted or is restricted from in that component. See also *authority*, *granted authority*, and *restricted authority*.

**action.** A task performed by the TeamConnection server and requested by a TeamConnection client. A TeamConnection action is the same as issuing one TeamConnection command.

**agent.** See *build agent*.

**alternate version ID.** In collision records, the name of a version of a driver, release, or work area where the conflicting version of a part is visible.

**approval record.** A status record on which an approver must give an opinion of the proposed part changes required to resolve a defect or implement a feature in a release.

**approver.** A user who has the authority to mark an approval record with accept, reject, or abstain within a specific release.

**approver list.** A list of user IDs attached to a release, representing the users who must review part changes that are required to resolve a defect or implement a feature in that release.

**attribute.** Information contained in a field that is accessible to the user. TeamConnection enables family administrators to customize defect, feature, user, and part tables by adding new attributes.

**authority.** The right to access development objects and perform TeamConnection commands. See also *access list*, *base authority*, *explicit authority*, *granted authority*, *implicit authority*, *restricted authority*, and *superuser privilege*.

## B

**base authority.** The set of actions granted to a user when a user ID is created within a TeamConnection family. See also *authority*. Contrast with *implicit authority* and *explicit authority*.

**base name.** The name assigned to the part outside of the TeamConnection server environment, excluding any directory names. See also *path name*.

**base part tree.** The base set of parts associated with a release, to which changes are applied over time. Each committed driver or work area for a release updates the base part tree for that release.

**build.** The process used to create applications within TeamConnection.

**build agent.** A program that handles access to persistent data on behalf of the build processor. Each build agent is connected to one and only one build processor, through a TCP/IP connection.

**build associate.** A TeamConnection part that is not an input to or an output from a build. An example of such a part is a read.me file.

**build cache.** A directory that the build processor uses to enhance performance.

**build dependent.** A TeamConnection part that is needed for the compile operation to complete, but it will not be passed directly to the compiler. An example of this is an include file. See also *dependencies*.

**builder.** An object that can transform one set of TeamConnection parts into another by invoking tools such as compilers and linkers.

**build event.** An individual step in the build of an application, such as the compiling of hello.c into hello.obj.

**build input.** A TeamConnection part that will be used as input to the object being built.

**build output.** A TeamConnection part that will be generated output from a build, such as an .obj or .exe file.

**build pool.** A group of build servers that resides in an environment. The environment in which several build servers operate. Typically, several servers are set up for each environment that the enterprise develops applications for.

**build processor.** A program that invokes the tools, such as compilers and linkers, that construct an application. Each build processor is connected to one and only one build agent, through a TCP/IP connection. See also *build agent* and *build cache*.

**build scope.** A collection of build events that implement a specific build request. See also *build event*.

**build script.** An executable or command file that specifies the steps that should occur during a build operation. This file can be a compiler, a linker, or the name of a .cmd file you have written.

**build server.** The combination of a build processor and a build agent. See also *build agent* and *build processor*.

**build target.** The name of the part at the top of the build tree which is the final output of a build. TeamConnection uses the build target to determine the scope of the build. See also *build tree*.

**build tree.** A graphical representation of the dependencies that the parts in an application have on one another. If you change the relationship of one part to another, the build tree changes accordingly.

## C

**change control process.** The process of limiting and auditing changes to parts through the mechanism of checking parts in and out of a central, controlled, storage location. Change control for individual releases can be integrated with problem tracking by specifying a process for the release that includes the tracking subprocess.

**check in.** The return of a TeamConnection part to version control.

**check out.** The retrieval of a version of a part under TeamConnection control. In non-concurrent releases, the check out operation does not allow a second user to check out a part until the first user has checked it back in.

**child component.** Any component in a TeamConnection family, except the root component, that is created in reference to an existing component. The existing component is the parent component, and the new component is the child component. A parent

component can have more than one child component, and a child component can have more than one parent component. See also *component* and *parent component*.

**child part.** Any part in a build tree that has a parent defined. A child part can be input, output, or dependent. See also *part* and *parent part*.

**client.** A functional unit that receives shared services from a server. Contrast with *server*.

**collision record.** A status record associated with a work area or driver, a part, and one of the following:

- The work area or driver's release
- Another work area

TeamConnection generates a collision record when a user attempts to replace an older version of a part with a modified version, another user has already modified that part, and the first user's modification is not based on this latest version of the part.

**command.** A request to perform an operation or run a program from the command line interface. In TeamConnection, a command consists of the command name, one action flag, and zero or more attribute flags.

**command line.** (1) An area on the Tasks window or in the TeamConnection Commands window where a user can type TeamConnection commands. (2) An area on an operating system window where you can type TeamConnection commands.

**committed version.** The revision of a part that is visible from the release.

**common part.** A part that is shared by two or more releases, and the same version of the part is the current version for those releases.

**comparison operator.** An operator used in comparison expressions. Comparison operators used in TeamConnection are > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to), and = (equal to).

**component.** A TeamConnection object that organizes project data into structured groups, and controls configuration management properties. Component owners can control access to data and notification of TeamConnection actions. Components exist in a parent-child hierarchy, with descendant components inheriting access and notification information from ancestor components. See also *access list* and *notification list*.

**concurrent development.** Several users can work on the same part at the same time. TeamConnection requires these users to reconcile their changes when they commit or integrate their work areas and drivers with the release. Contrast with *serial development*. See also *work area*.



**configuration management.** The process of identifying, managing, and controlling software modules as they change over time.

**connecting parts.** The process of linking parts so that they are included in a build.

**context.** The current work area or driver used for part operations.

**corequisite work areas.** Two or more work areas designated as corequisites by a user so that all work areas in the corequisite group must be included as members in the same driver, before that driver can be committed. If the driver process is not used in the release, then all corequisite work areas must be integrated by the same command. See also *prerequisite work areas*.

**current version.** The last visible modification of a part in a driver, release, or work area.

**current working directory.** (1) The directory that is the starting point for relative path names. (2) The directory in which you are working.

## D

**daemon.** A program that runs unattended to perform a standard service. Some daemons are triggered automatically to perform their task; others operate periodically.

**database.** A collection of data that can be accessed and operated upon by a data processing system for a specific purpose.

**default.** A value that is used when an alternative is not specified by the user.

**default query.** A database search, defined for a specific TeamConnection window, that is issued each time that TeamConnection window is opened. See also *search*.

**defect.** A TeamConnection object used to formally report a problem. The user who opens a defect is the defect originator.

**delete.** If you delete a development object, such as a part or a user ID, any reference to that object is removed from TeamConnection. Certain objects can be deleted only if certain criteria are met. Most objects that are deleted can be re-created.

**delta part tree.** A directory structure representing only the parts that were changed in a specified place.

**dependencies.** In TeamConnection builds there are two types of dependencies:

- **automatic.** These are build dependencies that a parser identifies.

- **manual.** These are build dependencies that a user explicitly identifies in a build tree.

See also *build dependent*.

**descendant.** If you descendant a development object, such as, a part or a user ID, any reference to that object is removed from TeamConnection. Certain objects can be descendant only if certain criteria are met. Most objects that are descendants can be re-created.

**disconnecting parts.** The process of unlinking parts so that they are not included in a build.

**driver.** A collection of work areas that represent a set of changed parts within a release. Drivers are only associated with releases whose processes include the track and driver subprocesses.

**driver member.** A work area that is added to a driver.

## E

**end user.** See *user*.

**environment.** (1) A user-defined testing domain for a particular release. (2) A defect field, in which case it is the environment where the problem occurred. (3) The string that matches a build agent with a build event.

**environment list.** A TeamConnection object used to specify environments in which a release should be tested. A list of environment-user ID pairs attached to a release, representing the user responsible for testing each environment. Only one tester can be identified for an environment.

**explicit authority.** The ability to perform an action against a TeamConnection object because you have been granted the authority to perform that action. Contrast with *base authority* and *implicit authority*.

**extract.** A TeamConnection action you can perform on a builder, part, driver or release builder. An extraction results in copying the specified builder, part, or parts contained in the driver or release to a client workstation.

## F

**family.** A logical organization of related data. A single TeamConnection server can support multiple families. The data in one family cannot be accessed from another family.

**family administrator.** A user who is responsible for all nonsystem-related tasks for one or more TeamConnection families, such as planning, configuring, and maintaining the TeamConnection environment and managing user access to those families.

**family server.** A workstation running the TeamConnection server software.

**FAT.** See *file allocation table*.

**feature.** A TeamConnection object used to formally request and record information about a functional addition or enhancement. The user who opens a feature is the feature originator.

**file.** A collection of data that is stored by the TeamConnection server and retrieved by a path name. Any text or binary file used in a development project can be created as a TeamConnection file. Examples include source code, executable programs, documentation, and test cases.

**file allocation table (FAT).** The DOS- and OS/2-compatible file system that manages input, output, and storage of files on your system. File names can be up to 8 characters long, followed by a file extension that can be up to 3 characters.

**fix record.** A status record that is associated with a work area and that is used to monitor the phases of change within each component that is affected by a defect or feature for a specific release.

**freeze.** The freeze action saves changed parts to the work area. Thus, TeamConnection takes a snapshot of the work area, including all of the current versions of parts visible from that work area, and saves this image of the system. The user can always come back to this stage of development in the work area. Note, however, that a freeze action does not make the changes visible to the other people working in the release. Compare with *refresh*.

**full part tree.** A directory structure representing a complete set of active parts associated with the release.

## G

**Gather.** A tool to organize files for distribution into a specified directory structure. This tool can be used as a prelude to further distribution, such as using CD-ROM or through electronic means like Netview DM/2. It can also be used by itself for distributing file copies to network-attached file systems.

**GID.** A number which uniquely identifies a file's group to an AIX system.

**granted authority.** If an authority is granted on an access list, then it applies for all objects managed by this component and any of its descendants for which the authority is not restricted. See also *access list*, *authority*, and *inheritance*. Contrast with *restricted authority*.

**graphical user interface (GUI).** A type of computer interface consisting of a visual metaphor of a real-world

scene, often as a desktop. Within that scene are icons, representing actual objects, that the user can access and manipulate with a pointing device.

**GUI.** Graphical user interface.

## H

**high-performance file system (HPFS).** In the OS/2 operating system, an installable file system that uses high-speed buffer storage, known as a cache, to provide fast access to large disk volumes. The file system also supports the existence of multiple, active file systems on a single personal computer, with the capacity of multiple and different storage devices. File names used with HPFS can have as many as 254 characters.

**host.** A host node, host computer, or host system.

**host list.** A list associated with each TeamConnection user ID that indicates the client machine that can access the family server and act on behalf of the user. The family server uses the list to authenticate the identity of a client machine when the family server receives a command. Each entry consists of a login, a host name, and a TeamConnection user ID.

**host name.** The identifier associated with the host computer.

**HPFS.** See *high-performance file system*.

## I

**implicit authority.** The ability to perform an action on a TeamConnection object without being granted explicit authority. This authority is automatically granted through inheritance or object ownership. Contrast with *base authority* and *explicit authority*.

**import.** To bring in data. In TeamConnection, to bring selected items into a field from a matching TeamConnection object window.

**inheritance.** The passing of configuration management properties from parent to child component. The configuration management properties that are inherited are access and notification. Inheritance within each TeamConnection family or component hierarchy is cumulative.

**integrated problem tracking.** The process of integrating problem tracking with change control to track all reported defects, all proposed features, and all subsequent changes to parts. See also *change control*.

**interest group.** The list of actions that trigger notification to the user IDs associated with those actions listed in the notification list.

## J

**job queue.** A queue of build scopes. One job queue exists for each TeamConnection family.

## L

**lock.** An action that prevents editing access to a part stored in the TeamConnection development environment so that only one user can change a part at a time.

**login.** The name that identifies a user on a multi-user system, such as AIX or HP-UX. In OS/2 and Windows, the login value is obtained from the TC\_USER environment variable.

## M

**map.** The process of reassigning the meaning of an object.

**metadata.** In databases, data that describe data objects.

## N

**name server.** In TCP/IP, a server program that supplies name-to-address translation by mapping domain names to Internet addresses.

**National Language Support (NLS).** The modification or conversion of a United States English product to conform to the requirements of another language or country. This can include the enabling or retrofitting of a product and the translation of nomenclature, MRI, or documentation of a product.

**Network File System (NFS).** The Network File System is a program that enables you to share files with other computers in networks over a variety of machine types and operating systems.

**notification list.** An object that enables component owners to configure notification. A list attached to a component that pairs a list of user IDs and a list of interest groups. It designates the users and the corresponding notification interest that they are being granted for all objects managed by this component or any of its descendants.

**notification server.** A server that sends notification messages to the client.

**NTFS.** NT file system.

**NVBridge.** A tool for automatic electronic distribution of TeamConnection software deliverables within a NetView DM/2 network.

## O

**operator.** A symbol that represents an operation to be done. See also *comparison operators*.

**originator.** The user who opens a defect or feature and is responsible for verifying the outcome of the defect or feature on a verification record. This responsibility can be reassigned.

**owner.** The user who is responsible for a TeamConnection object within a TeamConnection family, either because the user created the object or was assigned ownership of the object.

## P

**parent component.** All components in each TeamConnection family, except the root component, are created in reference to an existing component. The existing component is the parent component. See also *child component* and *component*.

**parent part.** Any part in a build tree that has a child defined. See also *part* and *child part*.

**parser.** A tool that can read a source file and report back a list of dependencies of that source file. It frees a developer from knowing the dependencies one part has on other parts to ensure a complete build is performed.

**part.** A collection of data that is stored by the family server and retrieved by a path name. They include text objects, binary objects, and modeled objects. These parts can be stored by the user or the tool, or they can be generated from other parts, such as when a linker generates an executable file.

**path name.** The name of the part under TeamConnection control. A path name can be a directory structure and a base name or just a base name. It must be unique within each release. See also *base name*.

**pool.** See *build pool*.

**pop-up menu.** A menu that, when requested, appears next to the object it is associated with.

**prerequisite work areas.** If a part is changed to resolve more than one defect or feature, the work area referenced by the first change is a prerequisite of the work area referenced by later changes. A work area is a prerequisite to another work area if:

- Part changes are checked in, but not committed, for the first work area.
- One or more of the same parts are checked out, changed, and checked in again for the second work area.

**problem tracking.** The process of tracking all reported defects through to resolution and all proposed features through to implementation.

**process.** A combination of TeamConnection subprocesses, configured by the family administrator, that controls the general movement of TeamConnection objects (defects, features, work areas, and drivers) from state to state within a component or release. See also *subprocess* and *state*.

## Q

**query.** A request for information from a database, for example, a search for all defects that are in the open state. See also *default query* and *search*.

## R

**raw format.** Information retrieved on the report command that has the vertical bar delimiter separating field information, and each line of output corresponds to one database record.

**refresh.** This TeamConnection action updates a work area with any changes from the release, and it also freezes the work area, if it is not already frozen.

**relative path name.** The name of a directory or a part expressed as a sequence of directories followed by a part name, beginning from the current directory.

**release.** A TeamConnection object defined by a user that contains all the parts that must be built, tested, and distributed as a single entity.

**restricted authority.** The limitation on a user's ability to perform certain actions at a specific component. Authority can be restricted by the superuser, the component owner, or a user with AccessRestrict authority. See also *authority*.

**root component.** The initial component that is created when a TeamConnection family is configured. All components in a TeamConnection family are descendants of the root component. Only the root component has no parent component. See also *component*, *child component*, and *parent component*.

## S

**search.** To scan one or more data elements of a set in a database to find elements that have certain properties.

**serial development.** While a user has parts checked out from a work area, no one else on the team can check out the part. The user develops new material without interacting with other developers on the project. TeamConnection provides the opportunity to hold the part until the user is sure that it integrates with the rest

of the application. Thus, the lock is not released until the work area as a whole is committed. Contrast with *concurrent development*. See also *work area*.

**server.** A workstation that performs a service for another workstation.

**shared part.** A part that is contained in two or more releases.

**shell script.** A series of commands combined in a file that carry out a function when the file is run.

**SID.** The name of a version of a driver, release, or work area.

**sizing record.** A status record created for each component-release pair affected by a proposed defect or feature. The sizing record owner must indicate whether the defect or feature affects the specified component-release pair and the approximate amount of work needed to resolve the defect or implement the feature within the specified component-release pair.

**stanza format.** Data output generated by the Report command in which each database record is a stanza. Each stanza line consists of a field and its corresponding values.

**state.** Work areas, drivers, features, and defects move through various states during their life cycles. The state of an object determines the actions that can be performed on it. See also *process* and *subprocess*.

**subprocess.** TeamConnection subprocesses govern the state changes for TeamConnection objects. The design, size, review (DSR) and verify subprocesses are configured for component processes. The track, approve, fix, driver, and test subprocesses are configured for release processes. See also *process* and *state*.

**superuser.** This privilege lets a user perform any action available in the TeamConnection family.

**system administrator.** A user who is responsible for all system-related tasks involving the TeamConnection server, such as installing, maintaining, and backing up the TeamConnection server and the database it uses.

## T

**task list.** The list of tasks displayed in the Tasks window. The user can customize this list to issue requests for information from the server. Tasks can be added, modified, or deleted from the lists.

**TCP/IP.** Transmission Control Protocol/Internet Protocol.

**TeamConnection client.** A workstation that connects to the TeamConnection server by a TCP/IP connection and that is running the TeamConnection client software.

**TeamConnection part.** A part that is stored by the TeamConnection server and retrieved by a path name, release, type, and work area. See also *part*, *common part*, and *type*.

**TeamConnection superuser.** See *superuser*.

**tester.** A user responsible for testing the resolution of a defect or the implementation of a feature for a specific driver of a release and recording the results on a test record.

**test record.** A status record used to record the outcome of an environment test performed for a resolved defect or an implemented feature in a specific driver of a release.

**track subprocess.** An attribute of a TeamConnection release process that specifies that the change control process for that release will be integrated with the problem tracking process.

**Transmission Control Protocol/Internet Protocol (TCP/IP).** A set of communications protocols that support peer-to-peer connectivity functions for both local and wide area networks.

**type.** All parts that are created through the TeamConnection GUI or on the command line will show up in reports with the type of TCPart as the part type. The TeamConnection GUI and command line can only check in, check out, and extract parts of the type TCPart.

**Note:** Parts created through an API can have other specified types. Refer to the *Commands Programming Reference* for more information.

## U

**user exit.** A user exit allows TeamConnection to call a user-defined program during the processing of TeamConnection transactions. User exits provide a means by which users can specify additional actions that should be performed before completing or proceeding with a TeamConnection action.

**user ID.** The identifier assigned by the system administrator to each TeamConnection user.

## V

**verification record.** A status record that the originator of a defect or a feature must mark before the defect or feature can move to the closed state. Originators use verification records to verify the resolution or implementation of the defect or feature they opened.

**version.** (1) A specific view of a driver, release, or work area. (2) A revision of a part.

**version control.** The storage of multiple versions of a single part along with information about each version.

**view.** An alternative and temporary representation of data from one or more tables.

## W

**work area.** An object in TeamConnection that you create and associate with a release. When the work area is created, you see the most current view of the release and all the parts that it contains. You can check out the parts in the work area, make modifications, and check them back into the work area. You can also test the modifications without integrating them. Other users are not aware of the changes that you make in the work area until you integrate the work area to the release. While you work on files in a work area, you do not see subsequent part changes in the release until you integrate or refresh your work area.

**working part.** The checked-out version of a TeamConnection part.

## Y

**year 2000 ready.** IBM VisualAge TeamConnection is Year 2000 ready. When used in accordance with its associated documentation, TeamConnection is capable of correctly processing, providing and/or receiving date data within and between the twentieth and twenty-first centuries, provided that all products (for example, hardware, software and firmware) used with the product properly exchange accurate date data with it.





---

# Index

## Special Characters

/Ft(dir) builder parameter 286

## A

access command  
    creating example 133  
access list 131  
actions  
    Show Authority Actions window 131  
    Show Interest Actions window 137  
    user exit parameters for 393  
Add Host window 126  
Add Notification window 137  
age utility, for defects and features 169  
AIX  
    hardware requirements  
        for build processor 20  
        for family server 20  
    preparing syslog file 23  
    preparing to install code 21  
    software requirements 21  
approval subprocess 117  
audit log  
    cleaning up 175  
    description of 170  
    example of 171  
    information contained in 171  
authorit.ld 371  
authority  
    basic  
        reloading 371  
        verify loading of 372  
    build 90, 302  
    example of granting 127  
    granting to users 131  
    inheritance of 132  
    instructions for granting and restricting 132  
    planning for 127  
    Remove Access window 133  
    restricting 128, 132  
    types of 128  
authority groups  
    Authority Groups Settings notebook 130  
    creating from command line 371  
    creating or changing 129  
    definition of 129  
    display list of 131  
    worksheet 479  
Authority Groups Settings page 130  
authority table 371  
automatic pruning of work areas 114

## B

base authority 128  
build action 6  
build administrator 13

build agent  
    accessing database remotely 89, 93  
    assigning to build pools 93  
    at work 317  
    description of 260  
    for OS/2  
        hardware requirements 55  
        software requirements 57  
    for Windows NT  
        hardware requirements 69  
    installing on separate machines 267  
    startup file, creating 94  
    stopping 96  
build directory 90  
build environment  
    supported by builder 279  
build event  
    criteria used to determine success 280  
    defining multiple outputs from one event 321  
    definition of 261  
    determining available agents 92  
    timeout setting 280  
    with VisualAge C++ templates 286  
build function  
    authority 90, 302  
    canceling a build 320  
    collector part 322  
        example of 322  
    concepts of 259  
    creating an MVS build processor 272  
    definition of 11  
    diagram showing physical structure of 259  
    features of 259  
    installing 267  
    installing an MVS build processor 272  
    keeping build output versions 114  
    monitoring build progress  
        using Build Progress window 318  
        using part -viewmsg command 318  
    object model 263  
    startup files, creating 94  
build mode 314  
build pool  
    assigning agents to 93  
    specifying when starting build 314  
build processor  
    AIX  
        hardware requirements 20, 43  
    at work 317  
    build directory 90  
    cache directory 90  
    creating for MVS 272  
    description of 260  
    installing on separate machines 267  
    OS/2  
        hardware requirements 55  
        software requirements 57  
    starting MVS 91

- build processor (*continued*)
  - startup file, creating 94
  - stopping 96
  - updating environment variables 272, 275
  - Windows NT
    - hardware requirements 69
- build scope
  - definition of 261
  - determining 315
- build scripts
  - at work 317
  - debug variable 283
  - definition of 261
  - for MVS
    - compile example 295
    - definition of 287
    - file name conversions 291
    - link example 298
    - steps for working with 277, 287
    - supported JCL syntax 293
    - writing 291
  - for OS/2 281
  - modifying contents of 284
  - testing 284
  - timeout of 280
  - writing 282
  - writing an executable file for 283
- build server
  - definition of 260
  - timeout setting 280
- build target 314
- build tree
  - creating 309
  - diagram showing build times 315
  - display of 313
  - example of 264
  - multiple outputs from single event 321
  - setting up for packaging
    - for other distribution tools 329
    - for the gather tool 326
    - for the NVBridge tool 328
    - setting up for packaging 326
  - versions of 264
  - working with 263
- builder
  - command
    - connecting builder to its parts 285
    - creating a builder 278, 284
    - extracting a builder 285
    - modifying builder contents 285
  - connecting to parts 285, 303
  - creating a null builder 279, 322
  - definition 261
  - for MVS
    - creating 287
    - naming 288
    - passing parameters to a build script 292
    - versions of 288
  - for OS/2
    - creating 277
    - environment supported 279

- builder (*continued*)
  - for OS/2 (*continued*)
    - naming 278
    - passing parameters to a build script 281
    - versions of 278
  - removing from a part 286
- building an application
  - a build agent at work 317
  - an example
    - adding job to job queue 317
    - build processors at work 317
    - build scripts at work 317
    - creating a build tree 309
    - creating builders and parsers 309
    - determining the build scope 315
    - extracting resulting executable 318
    - list of tasks 307
    - starting a build on the client 313
    - starting processors and agents 308
  - authority 90, 302
  - building all parts ignoring times 319
  - canceling a build 320
  - monitoring progress of build 318
  - preparing your parts 265
  - report of which parts will be built 320
  - running in spite of errors 319
  - with VisualAge C++ and templates 286

## C

- cache data set
  - attributes of 91
  - deleting 91
  - description of 91
- cache directory
  - controlling size of 90
  - description of 90
- CD-ROM file system, adding and mounting 24, 36
- change control 3
- check-in action 6
- check-out action 6
- chfield command
  - syntax 378
  - using 378
- CID installation
  - setting up for client users 81
- client
  - AIX
    - software requirements 21, 44
    - starting 30, 52
  - definition 5
  - HP
    - starting 42
  - OS/2
    - software requirements 57
    - starting 67
  - Windows 3.1
    - starting 80
  - Windows NT
    - software requirements 70



- collector part
  - using a null builder 279
- command line interface
  - configuring processes 373
  - creating authority groups 371
  - creating interest groups 372
  - starting family server 86
  - starting notification server 87
- commands
  - chfield 378
  - dbcreate 40, 64, 77
  - fhchdf 370
  - fhcirt 369
  - fhclauth 371
  - fhclcnfg 377
  - fhclintr 373
  - fhclproc 374
  - migcmvc 227
  - migt 196
  - notifyd 87
  - ping 22, 34, 46, 59, 72
  - sendmail 86, 102
  - tcadmin 100
  - tcleanu 175
  - tclicmon 247
  - teamagnt 93
  - teamcd 86
- component command
  - example of creating 120
- Component Processes Settings page 154
- components
  - creating 119
  - definition of 7
  - example of hierarchy 7
  - example of using processes 118
  - information stored about 7
  - list of processes 116
  - list of subprocesses 116
  - naming 111
  - organizing hierarchy 109
  - ownership 111
  - planning 109
  - planning processes for 116
  - processes, configuring 373
- comproc.ld 374
- concepts of
  - build function 259
  - TeamConnection 4
- concurrent development
  - difference from serial 113
- config.ld 375
- config.sys
  - changing 270
  - updating for build installation 271
  - updating for OS/2 installation 61
- config table
  - column descriptions 376
  - editing the config.ld file 375
  - modifying 375
  - reloading 377
  - verify loading of 377
- Configurable Fields for Defects Settings page 145
- Configurable Fields Settings page 143
- configurable processes, worksheet 488
- configuration management 3
- configuring
  - fields
    - changing field types 143
    - changing field values 142
    - creating or changing 145
    - deleting 144, 146
    - displaying properties of 147
  - processes
    - about 153
    - editing the .ld files 373, 374
    - reloading the config table 374
    - using the GUI 153
  - updating for build processor 272, 275
  - user exits 162
  - worksheet 488
- courier.cmd 361
- create action 6
- Create Builder 277
- Create Components window 119
- Create Parser 301
- Create Parts window 309
- Create Releases window 120
- Create User window 124
- D**
- daemon
  - number of 85
- database
  - backing up 178
  - changing a file database to a rawfs 188
  - controlling size of 114
  - copying 181
  - creating a rawfs 187
  - creating with dbcreate 40, 64, 77
  - creating with fhcirt 369
  - pinging host 186
  - restoring 182
  - size restriction 113
  - statistics 186
  - using a remote database 115
  - using multiple databases 115
  - verify pointers 186
- dbcreate command 40, 64, 77
- DEBUG 283, 284
- default component process 116
- defect.fmt 381
- defects
  - changing age of 169
  - definition of 9
- dependencies on a build
  - defining through parsers 261
  - viewing through a build tree 263
- development component process 116
- development mode
  - selecting serial or concurrent 113
- disk space, checking for 62, 73
- driver member
  - definition of 117

- driver subprocess 117
  - definition of 117
  - example of 119
  - how to use 119
- drivers
  - definition of 9
- dsrDefect subprocess 116
- dsrFeature subprocess 116

## E

- edit action 6
- emergency\_fix component process 116
- ENV=( ) 385
- environment variables 425
  - manually updating for build function 271
  - manually updating for OS/2 62
  - manually updating for Windows 75
  - setting 432
    - before invoking MVS build script 292
    - before invoking OS/2 build script 281
    - before invoking Windows NT build script 281
    - for AIX environment 26
    - for HP environment 26, 38, 49
    - for Solaris environment 49
  - updating for build processor 272, 275
  - used for trace 176
- error log 170
- errors 170
- examples of
  - audit log 171
  - build agents and processors on separate machines 267
  - build script for an MVS compile 295
  - build script for an MVS link 298
  - build tree 264
  - build tree showing build times 315
  - building an application 307
  - changing processes 118
  - client/server network 4
  - component hierarchy 7, 109, 110, 111
  - display of a build tree 313
  - driver subprocess 119
  - executable file for a build script 283
  - granting authority to users 127
  - linking releases 121
  - release-component relationship 112
  - report formats 382
  - showing part/release/component relationship 8
  - stanza report 147
  - table format 150
  - teamcpak command for Gather 333
  - teamcpak command for NVBridge 342, 362
  - user exit program 159
  - writing a build script 282
- explicit authority 128
- extract action 6
- extracting parts
  - resulting build executable 318

## F

- family
  - creating
    - using dbcreate command 40, 64, 77
    - using fhcirt command 369
    - using GUI 99
  - definition of 6
  - planning 99
  - verify connection to 22, 34, 46, 59, 72
- family administrator
  - responsibilities 13
  - tasks 369
    - changing report formats 381
    - configuring processes 373
    - copying configurable fields 380
    - creating a family 40, 64, 77, 369
    - creating an initial superuser 370
    - creating configurable fields 378, 379
    - creating or modifying authority groups 371
    - creating or modifying interest groups 372
    - defining configurable field types 375
    - editing authorit.Id 371
    - editing comproc.Id 374
    - editing interest.Id 372
    - editing relproc.Id 374
    - editing userExit 384
    - license monitoring 247
    - reloading configurable process tables 374
    - reloading the authority table 371
    - reloading the config table 377
    - reloading the interest table 373
    - setting up user exits 384
    - updating configurable fields 381
- family server
  - AIX
    - hardware requirements 20
    - installation tasks 25
    - installing component code 26
    - preparing to install code 21
    - software requirements 21
  - HP
    - hardware requirements 32
    - installation tasks 37
    - installing component code 37
    - preparing to install code 33
    - software requirements 33
  - OS/2
    - hardware requirements 55
    - installation tasks 61
    - installing component code 59
    - preparing to install code 58
    - software requirements 57
  - Solaris
    - hardware requirements 43
    - installation tasks 47
    - installing component code 48
    - preparing to install code 44
    - software requirements 44
- specifying daemons 85
- starting 86
- stopping 95

- family server (*continued*)
  - Windows NT
    - hardware requirements 69
    - installation tasks 73
    - installing component code 73
    - preparing to install code 71
- Family Settings page 100
- feature.fmt, editing 381
- features
  - changing age of 169
  - definition of 9
- fhbbuild, build directory 90
- fhbcbatch, cache directory 90
- fhbhag.\$\$\$, listing of files in cache directory 90
- fhbopars.cmd, sample command file 305
- fhchdf command 370
- fhcirt command 369
- fhclauth command 371
- fhclcnfg command 377
- fhclintr command 373
- fhclproc command 374
- fhpicat utility 355
- fhpiscat utility 355
- fhpmcat utility 357
- fhpobdel utility 353
- fhpobdif utility 354
- fhpobmon utility 353
- fhprqmon utility 358
- fhprqpur utility 358
- fhpstat utility 353
- fhptrpur utility 360
- fhptrver utility 359
- fhpuccat utility 356
- fhpverif utility 357
- field types
  - creating or changing 143
  - deleting 144
- fields
  - configurable 141
    - changing field types 143
    - changing field values 142
    - conditions of 144
    - copying from another family 380
    - creating, using command line 378
    - creating, using GUI 145
    - deleting 144, 146
    - displaying properties of 147
    - modifying, using command line 378
    - modifying, using GUI 145
    - properties of 379
- files
  - authorit.ld 371
  - comproc.ld 374
  - config.ld 375
  - config.sys
    - changing 270
    - updating for OS/2 61
  - defect.fmt 381
  - feature.fmt 381
  - fhbbuild 90
  - fhbcbatch 90

- files (*continued*)
  - fhbhag.\$\$\$ 90
  - interest.ld 372
  - part.fmt 381
  - relproc.ld 374
  - resp.dat 82
  - updating hosts file for build servers 269
  - updating hosts file for servers 22, 34, 45, 58
  - updating services file for build servers 22, 33, 45, 58, 269
  - user.fmt 381
  - userExit 384
- fix subprocess 117

## G

- Gather tool
  - explanation of 331
  - packaging file
    - example of syntax 334
    - keywords for 335
    - specifying 332, 362
    - syntax rules for 334
    - writing 334
  - teamcpak command 332
- GUI
  - client
    - connecting builder to its parts 285
    - connecting parser to parts 303
    - creating parsers 301
    - removing builder connection from parts 286
    - removing parser connection from parts 304
  - family administrator
    - to add user exit programs 162
    - to change authority groups 130
    - to change configurable field types 143
    - to change configurable processes 153
    - to change interest groups 135
    - to change report formats 148
    - to change table formats 150
    - to create family 99
    - to create or change configurable fields 145
    - to start family server 88
    - to start notification server 88
    - to stop family server 95
    - to stop notification server 95

## H

- hardware requirements
  - for AIX build processor 20, 43
  - for AIX family server 20, 43
  - for HP family server 32
  - for OS/2 build agent 55
  - for OS/2 build processor 55
  - for OS/2 family server 55
  - for Windows NT build agent 69
  - for Windows NT build processor 69
  - for Windows NT family server 69
- hierarchy
  - component example 7, 109, 110
  - component ownership example 111

- hierarchy (*continued*)
  - release-component relationship 112
- host command
  - creating example 127
- host lists
  - Add Host window 126
  - creating entries 126
  - planning for 126
- host-only security 104
- hosts file
  - updating for build servers 269
  - updating for servers 22, 34, 45, 58
- HP
  - hardware requirements for family server 32
  - preparing syslog file 35
  - preparing to install code 33
  - software requirements 33

**I**

- implicit authority 128
- information model 3
- installation
  - build, preparing for 269
  - build agents and processors on separate machines 267
  - build components 270, 272
  - build function 267
  - creating an MVS build processor 272
  - how to start 270
  - of AIX components
    - CD-ROM file system, adding 24
    - from a CD-ROM 26
    - from a tape 26
    - preparing for 21
    - preparing syslog file 23
    - tasks 25
    - TeamConnection components 26
    - updating services file 71
    - verifying 27
  - of HP components
    - preparing for 33
    - preparing syslog file 35
    - tasks 37
    - TeamConnection components 37
    - updating services file 22, 45
    - verifying 39
  - of OS/2 components
    - fails because no user ID defined 71
    - preparing for 58
    - tasks 61
    - TeamConnection components 59
    - updating hosts file 58
    - updating services file 33
    - verifying 64
  - of Solaris components
    - from a CD-ROM 48
    - from a tape 48
    - preparing for 44
    - preparing syslog file 46
    - tasks 47

- installation (*continued*)
  - of Solaris components (*continued*)
    - TeamConnection components 48
    - verifying 49
  - of Windows components
    - preparing for 71
    - tasks 73
    - TeamConnection components 73
    - verifying 74
  - preparing installation for client users 81
  - TCP/IP for build 269
- interest groups
  - creating
    - using command line 372
    - using GUI 135
  - definition of 135
  - display list of 137
  - Interest Groups window 135
  - worksheet 483
- Interest Groups Settings page 135
- interest.id 372
- interest table
  - reloading 373
  - verify loading of 373
- interfaces
  - description of 5
- IP address
  - defining in hosts file 22, 34, 45, 58
  - defining through domain name server 83

## J

- job queue
  - adding a job 317
  - definition of 261

## K

- keyword
  - ATTEMPTS 341, 349
  - CLIENTINTERVAL 341, 348
  - CORPID 345
  - DATA 335, 343, 363
  - ENTRY 350
  - EXITPOST 337
  - EXITPRIOR 337
  - EXITREPLACE 337
  - for Gather utility 334, 335
  - INSTALLDIR 346
  - INSTALLPGM 346
  - INSTALLS 341, 342, 350, 362
  - IPARMS 347
  - MAIL 340, 344, 361
  - NVGLOBALS 344, 364, 365, 366
  - package file 334
  - PACKAGEFORMAT 335, 344, 364
  - RULE 335
  - SENDINTERVAL 341, 349
  - SENDS 341, 351, 362
  - SOURCE 336
  - TARGET 337

keyword (*continued*)

TARGETROOT 335  
TEAMCSERV 345  
TEST 341, 342, 350  
UNINSTALLPGM 348

## L

LAN installation  
    preparing for 81  
LANG 425  
license monitoring 247

## M

mail exit routines 86, 102  
mail facility 85, 101  
maintenance component process 116  
migration  
    CMVC-to-TeamConnection  
        guidelines 213  
        preparation 213  
        requirements 212  
    commands 233  
    migcmvc 227  
    migtc 196  
    recovery 190, 226  
    TeamConnection version 1-to-TeamConnection  
        version 2  
            fine-grained data 190  
            guidelines 190  
            preparation 191  
            requirements 190, 212  
Modify Part Properties 285  
MVS  
    build cache data set 91  
    build script  
        definition of 287  
        file name conversions 291  
        for a compile 295  
        for a link 298  
    builder 287  
    creating a build processor 272  
    installing a build processor 272  
    modifying RUNPGM JCL 91  
    starting build processor 91  
    stopping build processor 96  
    supported JCL syntax 293  
    syntax for builds 293

## N

name server  
    adding IP address to 83  
naming  
    builders 278  
    components 111  
    releases 115  
network 4  
NLSPATH 425  
    setting for family server 62, 75

notification  
    Add Notification window 137  
    adding for users 136  
    instructions for adding 137  
    planning for 134  
    restricting 137  
    setting up mail facility 85, 101  
notification list 136  
notification server  
    starting 86  
    stopping 95  
notify command  
    example of 138  
notifyd command 87  
nvbridge.cmd 339  
NVBridge tool  
    checking if object exists in catalog 355  
    checking install history for object 353  
    checking status of NetView DM/2 components 353  
    cross-referencing differences 354  
    explanation of 339  
    NetView DM/2 output files 339  
    packaging file  
        example of syntax 343  
        keywords for 343  
        specifying 340  
        syntax rules for 343  
        writing 342  
    problem determination 351  
    removing information about a cataloged object 353  
    sample build script 339  
    teamcpak command 340  
    used as a builder for packaging 339  
    utilities 352

## O

ObjectStore  
    configuring for OS/2 63  
    configuring for Windows 75  
    database utilities 177  
    server requirements 99  
OS/2  
    hardware requirements for build agent 55  
    hardware requirements for build processor 55  
    hardware requirements for family server 55  
    preparing to install code 58  
    software requirements 57  
OS\_AS\_SIZE 425  
OS\_CACHE\_SIZE 425  
OS\_NETWORK 425  
    setting for build agent 271  
    setting for family server 62, 75  
OS\_ROOTDIR 425  
    setting for AIX environment 26  
    setting for build agent 271  
    setting for family server 62, 75  
    setting for Solaris environment 49  
OS\_TMPDIR 425  
    setting for build agent 271  
    setting for family server 62, 75

- osbackup utility 178
- oscp utility 181
- osrestor utility 182
- ossvrsta utility 186
- osverify utility 186
- osvrpin utility 186

## P

- package file
  - for Gather tool
    - keywords for 335
    - specifying 332, 362
    - syntax rules for 334
    - writing 334
  - for NVBridge tool
    - example of syntax 343
    - keywords for 343
    - specifying 340
    - syntax rules for 343
    - writing 342
  - for Tivoli Courier tool
    - example of syntax 363
    - keywords for 363
    - syntax rules for 363
    - writing 363
- packaging
  - definition of 11
  - explanation of Gather tool 331
  - NVBridge tool 339
  - setting up build tree
    - for other distribution tools 329
    - for the gather tool 326
    - for the NVBridge tool 328
  - tasks involved in 325, 361
- parameters
  - passing to a build script 281
  - where specified
    - attributes of a builder 281
    - attributes of part in a build tree 281
    - parameters of part -build command 282
- parser command
  - connecting parser to parts 304
  - creating a parser 302
- parsers
  - command file
    - fhbopars.cmd, sample shipped 305
    - fhbopars.cmd sample shipped 305
    - specifying 302
    - writing 305
  - creating 301
  - definition of 261
  - explanation of 301
  - removing from a part 304
- part command
  - canceling a build 320
  - extracting build executable 318
  - listing parts that will be built 320
  - monitoring progress of build 318
  - removing builder connection 286
  - removing parser connection 305
  - starting a build 314

- part command (*continued*)
  - touching a part 316
- part.fmt 381
- parts
  - connecting builder 285
  - connecting parser 303
  - definition 6
  - removing builder from 286
  - removing parser from 304
- passwords 65, 78, 103, 105, 125, 370
- PATH 425
- ping command 22, 34, 46, 59, 72
- planning
  - component hierarchy 109
  - for authority to access data 127
  - for host lists 126
  - for notification 134
  - for user access 127
  - for user IDs 112, 123
  - planning 115
- port address
  - identifying in services file 22, 33, 45
  - specifying for build servers 269
- preship component process 116
- processes
  - configuring 153
  - definition of 9
  - example of using 118
  - for components
    - definition of 116
    - shipped 116
    - subprocesses 116
  - for releases
    - definition of 116
    - shipped 117
    - subprocesses 117
  - planning 115
  - worksheet 488
- prototype component process 116
- pruning 114

## R

- raw file system (rawfs) database
  - changing a file database to a rawfs 188
  - creating 187
- release command
  - creating example 121
- release management 3
- releases
  - creating 120
  - creating from an old release 121
  - definition of 7
  - example of linking releases 121
  - example of relationship with other objects 8
  - example of using processes 118
  - list of processes 117
  - list of subprocesses 117
  - naming 115
  - planning 112
  - planning processes for 116
  - processes, configuring 373



- releases (*continued*)
  - selecting serial or concurrent development 113
  - using a remote database 115
  - using multiple databases 115
- relproc.ld 374
- Remove Access window 133
- reports
  - changing format of 147
    - editing .fmt files 381
    - using GUI 148, 150
  - description of format sections 381
  - example of format 382
  - table format example 150
- resetAge utility 170
- resp.dat file 81
- response file
  - default values 81
  - updating 82
- return codes 352
  - from user exit program 156
- root component
  - owner 111
- RUNPGM JCL 91

## S

- sample files shipped
  - build script for NVBridge tool 339, 361
  - mail exit routines 86, 102
- security 65, 78, 103, 104, 125, 370
- sendmail command 86, 102
- serial development
  - difference from concurrent 113
- servers
  - build, installing 270, 272
  - configuring ObjectStore for OS/2 63
  - configuring ObjectStore for Windows 75
  - definition 5
  - family server
    - definition 5
    - instructions for installing AIX code 26
    - instructions for installing HP code 37
    - instructions for installing OS/2 code 59
    - instructions for installing Solaris code 48
    - instructions for installing Windows code 73
    - specifying daemons 85
    - starting 86
  - notification server 86
- services file
  - updating for build servers 269
  - updating for family servers 22, 33, 45, 58
- Settings notebook
  - Family Settings page 100
- Show Authority Actions window 131
- Show Interest Actions window 137
- socket port
  - format for specifying for build agent 93
- software requirements
  - for AIX client 21
  - for AIX family server 21
  - for HP build agent 33
  - for HP family server 33
- software requirements (*continued*)
  - for OS/2 build agent 57
  - for OS/2 build processor 57
  - for OS/2 client 57
  - for OS/2 family server 57
  - for Solaris client 44
  - for Solaris family server 44
  - for Windows NT client 70
- Solaris
  - hardware requirements
    - for build processor 43
    - for family server 43
  - preparing syslog file 46
  - preparing to install code 44
  - software requirements 44
- stanza report
  - changing format of 147
  - example of 147
- Stanza View Format Settings page 148
- starting
  - a build on the client 313
  - family server 86
  - MVS build processor 91
  - notification server 86
- startup files 94
- stopping
  - build agent 96
  - MVS build processor 96
  - OS/2 build processor 96
- subprocesses
  - for components 116
  - for releases 117
  - using driver subprocess 119
- superuser
  - creating others 125
  - creating using fhchdf 370
  - definition of 6, 129
- syntax
  - for MVS builds 293
  - of NVBridge utilities 352
  - supported JCL syntax for build 293
- syslog file
  - preparing for AIX 23
  - preparing for HP 35
  - preparing for Solaris 46
- system administrator, responsibilities 13

## T

- table report
  - changing format of 150
  - displaying 150
  - example of 150
- Table View Format Settings page 150
- tasks
  - family administrator 107
  - initial installation of AIX components 25, 47
  - initial installation of HP components 37
  - initial installation of OS/2 components 61
  - initial installation of Windows components 73
  - preparing to build an application 265
- TC\_BATCH\_TXNS 425

- TC\_BECOME 425
  - setting for family server 62, 75
  - setting for superuser access 125
- TC\_BUILD\_AGENT\_NOISY 425
- TC\_BUILD\_AGENTS\_FILE 86, 87, 94, 425
- TC\_BUILD\_NOKEY 425
- TC\_BUILD\_PROC\_NOISY 425
- TC\_BUILD\_PROCESSORS\_FILE 86, 87, 94, 425
- TC\_BUILDPOOL 425
- TC\_BULKCHUNKSIZE 425
- TC\_CACHEPRUNEMETHOD 90, 92, 272, 275, 425
- TC\_CACHESIZELIMIT 90, 92, 272, 275, 425
- TC\_CASESENSE 425
- TC\_COMPONENT 425
- TC\_DBPATH 87, 89, 94, 370, 425
  - setting for build agent 271
  - setting for family server 62, 75
- TC\_DEADLOCK\_DEBUG 425
- TC\_DEADLOCKBEEP 425
- TC\_DECACHE\_COUNT 425
- TC\_DECACHE\_WINDOW 425
- TC\_FAMILY 281, 308, 425, 432
  - setting for family server 62, 75
- TC\_FAMILY\_SERVER\_NOISY 425
- TC\_INPUT 281, 283, 284
- TC\_INPUTTYPE 281
- TC\_JOB\_QUEUE\_INTERVAL 425
- TC\_LOCATION 281
- TC\_MAKEIMPORTRULES 425
- TC\_MAKEIMPORTTOP 425
- TC\_MAKEIMPORTVERBOSE 425
- TC\_MIGRATERULES 425
- TC\_NOTIFY\_DAEMON 87, 88, 425
- TC\_OUTPUT 283, 284
- TC\_OUTPUTTYPE 281
- TC\_RECYCLE\_DEADLOCK 425
- TC\_RECYCLE\_SERVICE\_COUNT 425
- TC\_RELEASE 281, 425
- TC\_STACK\_TRACE 425
- TC\_SYSTEM\_LOG 425
- TC\_TMP 425
- TC\_TOP 425
- TC\_TRACE 425
- TC\_TRACEATTEMPTS 425
- TC\_TRACEDELAY 425
- TC\_TRACEFILE 425
- TC\_TRACESIZE 425
- TC\_USER 425
  - setting for family server 62, 75
- TC\_WORKAREA 281, 425
- tcadmin command 100
- tcleanu command 175
- tclicmon command 247
- TCP/IP
  - adding IP address to name server 83
  - installing for build 269
  - OS/2
    - if no USER environment variable 71
    - updating files 71
  - sendmail command 86, 102
  - specifying socket port for build agent 93

- TCP/IP (*continued*)
  - updating files for build installation 269
- teamagt command 93
- teamcd command 86
- TeamConnection
  - concepts of 4
  - diagram showing physical structure of 259
  - introducing 3
- teamcpak command
  - for Gather tool
    - command line flags 332
    - examples of 333
    - starting 332
    - syntax of 332
  - for NVBridge tool
    - command line flags 341
    - examples of 342, 362
    - starting 340
    - syntax of 340
  - using NVBridge utilities instead 352
- templates 286
- test component process 116
- test subprocess 117
- Tivoli Courier tool
  - explanation of 361
  - packaging file
    - example of syntax 363
    - keywords for 363
    - syntax rules for 363
    - writing 363
  - problem determination 366
  - sample build script 361
  - teamcpak command 361
  - Tivoli Courier output files 361
  - used as a builder for packaging 361
- TMP 425
- touch action 316
- trace 176
- track subprocess 117

## U

- univos2 67
- univunix 29, 41, 51, 52
- univwin 74
- user command
  - creating example 125
- User Exit parameters settings page 164
- User Exit Settings page 162
- user exits
  - configuring 162
  - customizing parameters for 385
  - editing userExit file 384
  - example of 159
  - nonzero return codes 156
  - parameters of 393
  - tips for writing 155
- user.fmt 381
- user IDs
  - creating 124
  - initial ID cannot be created 71
  - initial superuser 370



- user IDs (*continued*)
  - planning for 123
- userExit file 155
- users
  - notification 134
  - preparing for 123
  - to have data access 127
- utilities
  - database
    - osbackup 178
    - oscp 181
    - osrestor 182
    - ossvrsta 186
    - osverify 186
    - osvrpin 186
  - NVBridge
    - displaying syntax of 352
    - fhpicat 355
    - fhpiscat 355
    - fhpmcat 357
    - fhpobdel 353
    - fhpobdif 354
    - fhpobmon 353
    - fhprqmon 358
    - fhprqpur 358
    - fhpstat 353
    - fhptrpur 360
    - fhptrver 359
    - fhpucat 356
    - fhpverif 357

## V

- verifyDefect subprocess 116
- verifyFeature subprocess 116
- version control 3
- versions
  - builders 278
  - of build trees 264
- VisualAge C++ 286

## W

- window examples
  - Add Host 126
  - Add Notification 137
  - Authority Groups Settings notebook 130
  - Component Processes Settings 154
  - Configurable Fields for Defects Settings 145
  - Configurable Fields Settings 143
  - Create Builder 277
  - Create Components 119
  - Create Parser 301
  - Create Parts 309
  - Create Releases 120
  - Create User 124
  - Family Settings 100
  - Interest Groups 135
  - Modify Part Properties 285
  - Remove Access 133
  - Show Authority Actions 131
  - Show Interest Actions 137

- window examples (*continued*)
  - Stanza View Format Settings 148
  - Table View Format Settings 150
  - User Exit parameters settings page 164
  - User Exit Settings 162
- Windows NT
  - hardware requirements for build agent 69
  - hardware requirements for family server 69
  - preparing to install code 71
  - software requirements 70
- work area
  - automatic pruning of 114
  - definition of 8
  - things you can do with 8







Part Number: 33H2571



Printed in the United States of America  
on recycled paper containing 10%  
recovered post-consumer fiber.

SC34-4551-01



33H2571

