



DB-Library™/C Reference Manual

## **Open Client™**

15.7

DOCUMENT ID: DC32600-01-1570-01

LAST REVISED: April 2012

Copyright © 2012 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase trademarks can be viewed at the Sybase trademarks page at <http://www.sybase.com/detail?id=1011207>. Sybase and the marks listed are trademarks of Sybase, Inc. ® indicates registration in the United States of America.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world.

Java and all Java-based marks are trademarks or registered trademarks of Oracle and/or its affiliates in the U.S. and other countries.

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names mentioned may be trademarks of the respective companies with which they are associated.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

# Contents

|                       |     |
|-----------------------|-----|
| About This Book ..... | xix |
|-----------------------|-----|

|                  |  |          |
|------------------|--|----------|
| <b>CHAPTER 1</b> | <b>Introducing DB-Library .....</b>                  | <b>1</b> |
|                  | Client/server architecture .....                     | 1        |
|                  | Types of clients .....                               | 2        |
|                  | Types of servers .....                               | 2        |
|                  | The Open Client and Open Server products .....       | 3        |
|                  | Open Client .....                                    | 3        |
|                  | Open Server .....                                    | 4        |
|                  | Open Client libraries .....                          | 4        |
|                  | What is in DB-Library/C? .....                       | 4        |
|                  | Comparing the library approach to Embedded SQL ..... | 5        |
|                  | Data structures for communicating with servers ..... | 6        |
|                  | DB-Library/C programming .....                       | 6        |
|                  | DB-Library/C datatypes .....                         | 11       |
|                  | DB-Library/C routines .....                          | 12       |
|                  | Initialization .....                                 | 13       |
|                  | Command processing .....                             | 15       |
|                  | Results processing .....                             | 16       |
|                  | Message and error handling .....                     | 22       |
|                  | Information retrieval .....                          | 24       |
|                  | Browse mode .....                                    | 26       |
|                  | Text and image handling .....                        | 28       |
|                  | Datatype conversion .....                            | 29       |
|                  | Process control flow .....                           | 30       |
|                  | Remote procedure call processing .....               | 30       |
|                  | Registered procedure call processing .....           | 31       |
|                  | Gateway passthrough routines .....                   | 33       |
|                  | Datetime and money .....                             | 34       |
|                  | Cleanup .....  | 35       |
|                  | Secure support .....                                 | 35       |
|                  | Miscellaneous routines .....                         | 35       |
|                  | Two-phase commit service special library .....       | 36       |
|                  | MIT Kerberos on DB-Library .....                     | 36       |

|                       |    |
|-----------------------|----|
| Sample programs ..... | 37 |
|-----------------------|----|

## CHAPTER 2

|                       |           |
|-----------------------|-----------|
| <b>Routines .....</b> | <b>39</b> |
| db12hour .....        | 48        |
| dbadata .....         | 49        |
| dbadlen .....         | 52        |
| dbaltbind .....       | 54        |
| dbaltbind_ps .....    | 59        |
| dbaltcolid .....      | 65        |
| dbaltlen .....        | 66        |
| dbaltop .....         | 67        |
| dbalttype .....       | 68        |
| dbaltutype .....      | 69        |
| dbanullbind .....     | 70        |
| dbbind .....          | 72        |
| dbbind_ps .....       | 77        |
| dbbufsize .....       | 82        |
| dbbylist .....        | 83        |
| dbcancel .....        | 84        |
| dbcquery .....        | 85        |
| dbchange .....        | 86        |
| dbcharsetconv .....   | 87        |
| dbcclose .....        | 88        |
| dbclrbuf .....        | 88        |
| dbclropt .....        | 89        |
| dbcmd .....           | 91        |
| DBCMDROW .....        | 92        |
| dbccolbrowse .....    | 93        |
| dbcollen .....        | 94        |
| dbccolname .....      | 95        |
| dbccolsource .....    | 97        |
| dbccoltype .....      | 98        |
| dbccoltypeinfo .....  | 99        |
| dbcolutype .....      | 100       |
| dbconvert .....       | 102       |
| dbconvert_ps .....    | 106       |
| DBCOUNT .....         | 112       |
| DBCURCMD .....        | 113       |
| DBCURROW .....        | 114       |
| dbcursor .....        | 115       |
| dbcursorbind .....    | 117       |
| dbcursorclose .....   | 119       |
| dbcursorcolinfo ..... | 120       |
| dbcursorfetch .....   | 121       |



|                     |     |
|---------------------|-----|
| dbcursorinfo .....  | 123 |
| dbcursoropen .....  | 124 |
| dbdata .....        | 128 |
| dbdate4cmp .....    | 129 |
| dbdate4zero .....   | 130 |
| dbdatechar .....    | 131 |
| dbdatecmp .....     | 132 |
| dbdatecrack .....   | 133 |
| dbdatetime .....    | 135 |
| dbdateorder .....   | 138 |
| dbdatepart .....    | 139 |
| dbdatezero .....    | 140 |
| dbdatlen .....      | 141 |
| dbdayname .....     | 142 |
| DBDEAD .....        | 143 |
| dberrhandle .....   | 144 |
| dbexit .....        | 148 |
| dbfcmd .....        | 149 |
| DBFIRSTROW .....    | 152 |
| dbfree_xlate .....  | 153 |
| dbfreebuf .....     | 154 |
| dbfreeequal .....   | 155 |
| dbfreesort .....    | 155 |
| dbgetchar .....     | 157 |
| dbgetcharset .....  | 157 |
| dbgetloginf .....   | 158 |
| dbgetusername ..... | 160 |
| dbgetmaxprocs ..... | 161 |
| dbgetnatlang .....  | 162 |
| dbgetoff .....      | 162 |
| dbgetpacket .....   | 164 |
| dbgetrow .....      | 165 |
| DBGETTIME .....     | 167 |
| dbgetuserdata ..... | 167 |
| dbhasretstat .....  | 168 |
| dbinit .....        | 170 |
| DBIORDESC .....     | 170 |
| DBIOWDESC .....     | 172 |
| DBISAVAIL .....     | 173 |
| dbisopt .....       | 173 |
| DBLASTROW .....     | 174 |
| dbload_xlate .....  | 175 |
| dbloadsort .....    | 176 |
| dblogin .....       | 177 |

|                    |     |
|--------------------|-----|
| dbloginfree .....  | 179 |
| dbmny4add .....    | 179 |
| dbmny4cmp.....     | 180 |
| dbmny4copy.....    | 181 |
| dbmny4divide.....  | 182 |
| dbmny4minus.....   | 183 |
| dbmny4mul.....     | 184 |
| dbmny4sub.....     | 185 |
| dbmny4zero .....   | 186 |
| dbmnyadd.....      | 187 |
| dbmnycmp.....      | 188 |
| dbmnycopy.....     | 189 |
| dbmnydec.....      | 190 |
| dbmnydivide.....   | 191 |
| dbmnydown.....     | 192 |
| dbmnyinc.....      | 194 |
| dbmnyinit.....     | 194 |
| dbmnymaxneg.....   | 196 |
| dbmnymaxpos.....   | 197 |
| dbmnyminus.....    | 198 |
| dbmnymul.....      | 199 |
| dbmnyndigit.....   | 200 |
| dbmnyscale .....   | 206 |
| dbmnysub.....      | 208 |
| dbmnyzero .....    | 209 |
| dbmonthname .....  | 209 |
| DBMORECMDS.....    | 210 |
| dbmoretext .....   | 211 |
| dbmsghandle.....   | 212 |
| dbname .....       | 216 |
| dbnextrow.....     | 217 |
| dbnpcreate .....   | 219 |
| dbnpdefine .....   | 222 |
| dbnullbind.....    | 224 |
| dbnumalts.....     | 225 |
| dbnumcols.....     | 225 |
| dbnumcompute ..... | 227 |
| DBNUMORDERS.....   | 227 |
| dbnumrets .....    | 228 |
| dbopen .....       | 229 |
| dbordercol .....   | 233 |
| dbpoll.....        | 234 |
| dbprhead.....      | 239 |
| dbprrow .....      | 240 |

|                       |     |
|-----------------------|-----|
| dbprtype .....        | 241 |
| dbqual .....          | 242 |
| DBRBUF .....          | 246 |
| dbreadpage .....      | 247 |
| dbreadtext .....      | 248 |
| dbrectos .....        | 250 |
| dbrecvpassthru .....  | 251 |
| dbregdrop .....       | 253 |
| dbregexec .....       | 254 |
| dbreghandle .....     | 256 |
| dbreginit .....       | 260 |
| dbreglist .....       | 262 |
| dbregnowatch .....    | 263 |
| dbregparam .....      | 265 |
| dbregwatch .....      | 269 |
| dbregwatchlist .....  | 274 |
| dbresults .....       | 275 |
| dbretdata .....       | 278 |
| dbretlen .....        | 282 |
| dbretname .....       | 283 |
| dbretstatus .....     | 285 |
| dbrettype .....       | 287 |
| DBROWS .....          | 289 |
| DBROWTYPE .....       | 289 |
| dbrpcinit .....       | 290 |
| dbrpcparam .....      | 292 |
| dbrpcsend .....       | 294 |
| dbrpwclr .....        | 295 |
| dbrpwset .....        | 296 |
| dbsafestr .....       | 297 |
| dbsechandle .....     | 299 |
| dbsendpassthru .....  | 303 |
| dbservcharset .....   | 305 |
| dbsetavail .....      | 306 |
| dbsetbusy .....       | 306 |
| dbsetconnect .....    | 309 |
| dbsetdefcharset ..... | 310 |
| dbsetdeflang .....    | 311 |
| dbsetidle .....       | 312 |
| dbsetifile .....      | 313 |
| dbsetinterrupt .....  | 314 |
| DBSETLAPP .....       | 317 |
| DBSETLCHARSET .....   | 318 |
| DBSETLENCRYPT .....   | 319 |

|                             |     |
|-----------------------------|-----|
| DBSETLHOST .....            | 320 |
| DBSETLMUTUALAUTH.....       | 321 |
| DBSETLNATLANG .....         | 322 |
| DBSETLNETWORKAUTH .....     | 322 |
| dbsetloginfo .....          | 323 |
| dbsetlogintime .....        | 325 |
| DBSETLPACKET .....          | 326 |
| DBSETLPWD.....              | 327 |
| DBSETLSERVERPRINCIPAL ..... | 328 |
| DBSETLUSER .....            | 329 |
| dbsetmaxprocs.....          | 329 |
| dbsetnull .....             | 330 |
| dbsetopt .....              | 332 |
| dbsetrow.....               | 334 |
| dbsettime.....              | 336 |
| dbsetuserdata .....         | 336 |
| dbsetversion.....           | 339 |
| dbspid.....                 | 340 |
| dbspr1row .....             | 341 |
| dbspr1rowlen.....           | 343 |
| dbsprhead .....             | 344 |
| dbsprline.....              | 346 |
| dbsqlexec .....             | 347 |
| dbsqllok.....               | 349 |
| dbsqlsend .....             | 354 |
| dbstrbuild.....             | 355 |
| dbstrcmp .....              | 358 |
| dbstrcpy.....               | 359 |
| dbstrlen .....              | 361 |
| dbstrsort .....             | 362 |
| dbtabbrowse.....            | 363 |
| dbtabcount .....            | 364 |
| dbtabname .....             | 365 |
| dbtabsource .....           | 366 |
| DBTDS .....                 | 368 |
| dbtextsize .....            | 368 |
| dbtsnewlen .....            | 369 |
| dbtsnewval .....            | 370 |
| dbtsput .....               | 371 |
| dbtxptr .....               | 372 |
| dbtxtimestamp .....         | 374 |
| dbtxtsnewval .....          | 375 |
| dbtxtsput.....              | 375 |
| dbuse .....                 | 376 |

|                     |     |
|---------------------|-----|
| dbvarylen.....      | 377 |
| dbversion.....      | 378 |
| dbwillconvert ..... | 379 |
| dbwritepage.....    | 381 |
| dbwritetext.....    | 382 |
| dbxlate.....        | 387 |
| Errors .....        | 389 |
| Options.....        | 407 |
| Types .....         | 412 |

## CHAPTER 3

|  |            |
|--|------------|
| <b>Bulk Copy Routines.....</b>                             | <b>417</b> |
| Introduction to bulk copy .....                            | 417        |
| Transferring data into the database .....                  | 417        |
| Transferring data out of the database to a flat file ..... | 419        |
| List of bulk copy routines.....                            | 420        |
| bcp_batch.....   | 421        |
| bcp_bind.....  | 422        |
| bcp_colfmt.....  | 426        |
| bcp_colfmt_ps .....  | 429        |
| bcp_colln .....  | 434        |
| bcp_colptr.....  | 435        |
| bcp_columns .....  | 435        |
| bcp_control.....   | 436        |
| bcp_done .....   | 439        |
| bcp_exec.....  | 439        |
| bcp_getl.....  | 441        |
| bcp_init.....  | 441        |
| bcp_moretext .....   | 444        |
| bcp_options.....   | 447        |
| bcp_readfmt .....  | 448        |
| bcp_sendrow .....  | 448        |
| BCP_SETL.....  | 450        |
| bcp_setxlate .....   | 450        |
| bcp_writfmt.....   | 451        |

## CHAPTER 4

|  |            |
|--|------------|
| <b>Two-Phase Commit Service.....</b>                 | <b>453</b> |
| Programming distributed transactions.....            | 453        |
| The commit service and the application program ..... | 454        |
| The probe process .....                              | 456        |
| Two-phase commit routines .....                      | 456        |
| Specifying the commit server .....                   | 457        |
| Two-phase commit sample program .....                | 458        |
| Program notes.....                                   | 464        |

|                         |     |
|-------------------------|-----|
| Program note 1.....     | 464 |
| Program note 2.....     | 464 |
| Program note 3.....     | 465 |
| Program note 4.....     | 465 |
| Program note 5.....     | 466 |
| Program note 6.....     | 466 |
| Program note 7.....     | 467 |
| Program note 8.....     | 467 |
| abort_xact .....        | 468 |
| build_xact_string ..... | 468 |
| close_commit .....      | 469 |
| commit_xact .....       | 470 |
| open_commit.....        | 470 |
| remove_xact.....        | 471 |
| scan_xact .....         | 472 |
| start_xact.....         | 472 |
| stat_xact.....          | 473 |

## APPENDIX A

|  |            |
|--|------------|
| <b>Cursors .....</b>   | <b>475</b> |
| Cursor overview .....  | 475        |
| DB-Library cursor capability .....                           | 475        |
| Differences between DB-Library cursors and browse mode ..    | 476        |
| Differences between DB-Library and Client-Library cursors .. | 476        |
| Sensitivity to change .....                                  | 477        |
| Static cursor .....  | 478        |
| Keyset-driven cursor .....                                   | 478        |
| Dynamic cursor .....   | 479        |
| Concurrency control .....                                    | 479        |
| DB-Library cursor functions.....                             | 480        |
| Holding locks .....  | 480        |
| Stored procedures used by DB-Library cursors .....           | 481        |

## APPENDIX B

|  |            |
|--|------------|
| <b>DB-Library Error Messages .....</b> | <b>483</b> |
| 20001 .....                            | 483        |
| 20002 .....                            | 483        |
| 20003 .....                            | 483        |
| 20004 .....                            | 484        |
| 20005 .....                            | 484        |
| 20006 .....                            | 484        |
| 20008 .....                            | 485        |
| 20009 .....                            | 485        |
| 20010 .....                            | 485        |
| 20011 .....                            | 486        |

---

|            |     |
|------------|-----|
| 20012..... | 486 |
| 20013..... | 486 |
| 20014..... | 487 |
| 20015..... | 487 |
| 20016..... | 487 |
| 20017..... | 488 |
| 20018..... | 488 |
| 20019..... | 488 |
| 20020..... | 489 |
| 20021..... | 489 |
| 20022..... | 490 |
| 20023..... | 490 |
| 20024..... | 490 |
| 20025..... | 491 |
| 20026..... | 491 |
| 20027..... | 491 |
| 20028..... | 492 |
| 20029..... | 492 |
| 20030..... | 492 |
| 20031..... | 493 |
| 20033..... | 493 |
| 20034..... | 493 |
| 20035..... | 494 |
| 20036..... | 494 |
| 20037..... | 494 |
| 20038..... | 495 |
| 20039..... | 495 |
| 20040..... | 495 |
| 20041..... | 496 |
| 20042..... | 496 |
| 20043..... | 496 |
| 20044..... | 497 |
| 20045..... | 497 |
| 20046..... | 497 |
| 20047..... | 498 |
| 20048..... | 498 |
| 20049..... | 498 |
| 20050..... | 499 |
| 20051..... | 499 |
| 20052..... | 499 |
| 20053..... | 500 |
| 20054..... | 500 |
| 20055..... | 500 |
| 20056..... | 501 |

---

|            |     |
|------------|-----|
| 20060..... | 501 |
| 20061..... | 501 |
| 20062..... | 502 |
| 20063..... | 502 |
| 20064..... | 502 |
| 20065..... | 503 |
| 20066..... | 503 |
| 20067..... | 503 |
| 20068..... | 504 |
| 20069..... | 504 |
| 20070..... | 504 |
| 20071..... | 505 |
| 20072..... | 505 |
| 20073..... | 505 |
| 20074..... | 506 |
| 20075..... | 506 |
| 20076..... | 506 |
| 20077..... | 507 |
| 20078..... | 507 |
| 20079..... | 507 |
| 20080..... | 508 |
| 20081..... | 508 |
| 20082..... | 508 |
| 20083..... | 509 |
| 20084..... | 509 |
| 20085..... | 509 |
| 20086..... | 510 |
| 20087..... | 510 |
| 20088..... | 510 |
| 20091..... | 511 |
| 20092..... | 511 |
| 20093..... | 512 |
| 20094..... | 512 |
| 20095..... | 512 |
| 20096..... | 513 |
| 20097..... | 513 |
| 20098..... | 513 |
| 20099..... | 514 |
| 20100..... | 514 |
| 20101..... | 514 |
| 20102..... | 515 |
| 20103..... | 515 |
| 20104..... | 515 |
| 20105..... | 516 |



---

|            |     |
|------------|-----|
| 20106..... | 516 |
| 20107..... | 516 |
| 20108..... | 517 |
| 20109..... | 517 |
| 20110..... | 517 |
| 20111..... | 518 |
| 20112..... | 518 |
| 20113..... | 518 |
| 20114..... | 519 |
| 20115..... | 519 |
| 20116..... | 519 |
| 20117..... | 520 |
| 20118..... | 520 |
| 20119..... | 521 |
| 20120..... | 521 |
| 20121..... | 521 |
| 20122..... | 522 |
| 20123..... | 522 |
| 20124..... | 522 |
| 20125..... | 523 |
| 20126..... | 523 |
| 20127..... | 523 |
| 20128..... | 524 |
| 20129..... | 524 |
| 20130..... | 524 |
| 20131..... | 525 |
| 20132..... | 525 |
| 20133..... | 525 |
| 20134..... | 526 |
| 20135..... | 526 |
| 20136..... | 526 |
| 20137..... | 527 |
| 20138..... | 527 |
| 20139..... | 527 |
| 20140..... | 528 |
| 20141..... | 528 |
| 20142..... | 528 |
| 20143..... | 529 |
| 20144..... | 529 |
| 20145..... | 529 |
| 20146..... | 530 |
| 20147..... | 530 |
| 20148..... | 530 |
| 20149..... | 531 |

---

|            |     |
|------------|-----|
| 20150..... | 531 |
| 20151..... | 531 |
| 20152..... | 532 |
| 20153..... | 532 |
| 20154..... | 532 |
| 20155..... | 533 |
| 20156..... | 533 |
| 20157..... | 533 |
| 20158..... | 534 |
| 20159..... | 534 |
| 20160..... | 534 |
| 20161..... | 535 |
| 20162..... | 535 |
| 20163..... | 535 |
| 20164..... | 536 |
| 20165..... | 536 |
| 20166..... | 536 |
| 20167..... | 537 |
| 20168..... | 537 |
| 20169..... | 537 |
| 20170..... | 538 |
| 20171..... | 538 |
| 20172..... | 538 |
| 20173..... | 539 |
| 20174..... | 539 |
| 20175..... | 539 |
| 20176..... | 540 |
| 20177..... | 540 |
| 20178..... | 540 |
| 20179..... | 541 |
| 20180..... | 541 |
| 20181..... | 541 |
| 20182..... | 542 |
| 20183..... | 542 |
| 20184..... | 542 |
| 20185..... | 543 |
| 20186..... | 543 |
| 20187..... | 543 |
| 20188..... | 544 |
| 20189..... | 544 |
| 20190..... | 544 |
| 20191..... | 545 |
| 20192..... | 545 |
| 20193..... | 545 |

---

|            |     |
|------------|-----|
| 20194..... | 546 |
| 20195..... | 546 |
| 20196..... | 546 |
| 20197..... | 547 |
| 20198..... | 547 |
| 20199..... | 547 |
| 20200..... | 548 |
| 20201..... | 548 |
| 20202..... | 548 |
| 20203..... | 549 |
| 20204..... | 549 |
| 20205..... | 549 |
| 20206..... | 550 |
| 20207..... | 550 |
| 20208..... | 550 |
| 20209..... | 551 |
| 20210..... | 551 |
| 20211..... | 551 |
| 20212..... | 552 |
| 20213..... | 552 |
| 20214..... | 552 |
| 20215..... | 553 |
| 20216..... | 553 |
| 20217..... | 553 |
| 20218..... | 554 |
| 20219..... | 554 |
| 20220..... | 554 |
| 20221..... | 555 |
| 20222..... | 555 |
| 20223..... | 555 |
| 20224..... | 556 |
| 20225..... | 556 |
| 20226..... | 556 |
| 20227..... | 557 |
| 20228..... | 557 |
| 20229..... | 557 |
| 20230..... | 558 |
| 20231..... | 558 |
| 20232..... | 558 |
| 20233..... | 559 |
| 20234..... | 559 |
| 20235..... | 559 |
| 20236..... | 560 |
| 20237..... | 560 |

---

|            |     |
|------------|-----|
| 20238..... | 560 |
| 20239..... | 561 |
| 20240..... | 561 |
| 20241..... | 561 |
| 20242..... | 562 |
| 20243..... | 562 |
| 20244..... | 562 |
| 20245..... | 563 |
| 20246..... | 563 |
| 20247..... | 564 |
| 20248..... | 564 |
| 20249..... | 564 |
| 20250..... | 565 |
| 20251..... | 565 |
| 20252..... | 565 |
| 20253..... | 566 |
| 20254..... | 566 |
| 20255..... | 566 |
| 20256..... | 567 |
| 20257..... | 567 |
| 20258..... | 567 |
| 20259..... | 568 |
| 20260..... | 568 |
| 20261..... | 568 |
| 20262..... | 569 |
| 20263..... | 569 |
| 20264..... | 569 |
| 20265..... | 570 |
| 20266..... | 570 |
| 20267..... | 571 |
| 20268..... | 571 |
| 20269..... | 571 |
| 20270..... | 572 |
| 20271..... | 572 |
| 20272..... | 572 |
| 20273..... | 573 |
| 20274..... | 573 |
| 20275..... | 573 |
| 20276..... | 574 |
| 20277..... | 574 |
| 20278..... | 574 |
| 20279..... | 575 |
| 20280..... | 575 |
| 20282..... | 575 |

---

|                   |            |
|-------------------|------------|
| 20283.....        | 576        |
| 20284.....        | 576        |
| 20285.....        | 576        |
| 20286.....        | 577        |
| 20287.....        | 577        |
| 20288.....        | 577        |
| 20289.....        | 578        |
| 20290.....        | 578        |
| 20291.....        | 578        |
| 20292.....        | 579        |
| 20293.....        | 579        |
| 20294.....        | 579        |
| 20295.....        | 580        |
| 20296.....        | 580        |
| 20297.....        | 580        |
| 20298.....        | 581        |
| 20299.....        | 581        |
| 20300.....        | 581        |
| 20301.....        | 582        |
| 20302.....        | 582        |
| <b>Index.....</b> | <b>583</b> |



# About This Book

This book contains reference information for the C version of Open Client™ DB-Library™.

## Audience

This book is intended to serve as a reference manual for programmers who are writing DB-Library applications. It is written for application programmers familiar with the C programming language.

## How to use this book

This book contains these chapters:

- Chapter 1, “Introducing DB-Library,” contains a brief introduction to DB-Library.
- Chapter 2, “Routines,” contains specific information about each DB-Library routine, such as what parameters the routine takes and what it returns.
- Chapter 3, “Bulk Copy Routines,” contains an introduction to bulk copy and specific information about each bulk copy routine.
- Chapter 4, “Two-Phase Commit Service,” contains a brief description of two-phase commit service and specific information about each two-phase commit service routine.
- Appendix A, “Cursors,” introduces DB-Library’s cursor routines.
- Appendix B, “DB-Library Error Messages,” contains information about DB-Library error messages.

## Related documents

You can see these books for more information:

- *The Open Server and SDK New Features for Windows, Linux, and UNIX*, which describes new features available for Open Server and the Software Developer’s Kit. This document is revised to include new features as they become available.
- *The Open Server Release Bulletin* for your platform contains important last-minute information about Open Server.
- *The Software Developer’s Kit Release Bulletin* for your platform contains important last-minute information about Open Client™ and SDK.

- 
- The *jConnect™ for JDBC™ Release Bulletin* contains important last-minute information about jConnect.
  - The *Open Client and Open Server Configuration Guide* for your platform contains information about configuring your system to run Open Client and Open Server.
  - The *Open Client Client-Library/C Programmers Guide* contains information on how to design and implement Client-Library applications.
  - The *Open Client Client-Library/C Reference Manual* contains reference information for Open Client Client-Library™.
  - The *Open Server Server-Library/C Reference Manual* contains reference information for Open Server Server-Library.
  - The *Open Client and Open Server Common Libraries Reference Manual* contains reference information for CS-Library, which is a collection of utility routines that are useful in both Client-Library and Server-Library applications.
  - The *Open Client and Open Server Programmers Supplement* for your platform contains platform-specific information for programmers using Open Client and Open Server. This document includes information about:
    - Compiling and linking an application
    - The sample programs that are included with Open Client and Open Server
    - Routines that have platform-specific behaviors
  - The *Installation and Release Bulletin Sybase® SDK DB-Library Kerberos Authentication Option* contains information about installing and enabling the MIT Kerberos security mechanism to be used on DB-Library. DB-Library only supports network authentication and mutual authentication in the Kerberos security mechanism.
  - The *Open Client and Open Server International Developers Guide* provides information about creating internationalized and localized applications.
  - The *Open Client Embedded SQL™/C Programmers Guide* explains how to use Embedded SQL and the Embedded SQL precompiler with C applications.
  - The *Open Client Embedded SQL™/COBOL Programmers Guide* explains how to use Embedded SQL and the Embedded SQL precompiler with COBOL applications.



- The *jConnect for JDBC Programmers Reference* describes the jConnect for JDBC product and explains how to access data stored in relational database management systems.
- The *Adaptive Server® Enterprise ADO.NET Data Provider Users Guide* provides information on how to access data in Adaptive Server using any language supported by .NET, such as C#, Visual Basic .NET, C++ with managed extension, and J#.
- The *Adaptive Server Enterprise ODBC Driver by Sybase® Users Guide* for Microsoft Windows and UNIX, provides information on how to access data from Adaptive Server on Microsoft Windows and UNIX platforms, using the Open Database Connectivity (ODBC) Driver.
- The *Adaptive Server Enterprise OLE DB Provider by Sybase Users Guide for Microsoft Windows* provides information on how to access data from Adaptive Server on Microsoft Windows platforms, using the Adaptive Server OLE DB Provider.
- The *Adaptive Server Enterprise Database Driver for Perl Programmers Guide* provides information for Perl developers to connect to an Adaptive Server database and query or change information using a Perl script.
- The *Adaptive Server Enterprise extension module for PHP Programmers Guide* provides information for PHP developers to execute queries against an Adaptive Server database.
- The *Adaptive Server Enterprise extension module for Python Programmers Guide* provides information about Sybase-specific Python interface that can be used to execute queries against an Adaptive Server database.

**Other sources of information**

Use the Sybase Product Documentation Web site to learn more about your product:

- The Sybase Product Documentation Web site is accessible using a standard Web browser. In addition to product documentation, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

To access the Sybase Product Documentation Web site, go to Product Documentation at <http://www.sybase.com/support/manuals/>.

**Sybase certifications on the Web**

Technical documentation at the Sybase Web site is updated frequently.

---

❖ **Finding the latest information on product certifications**

- 1 Point your Web browser to Technical Documents at <http://www.sybase.com/support/techdocs/>.
- 2 Click Partner Certification Report.
- 3 In the Partner Certification Report filter select a product, platform, and timeframe and then click Go.
- 4 Click a Partner Certification Report title to display the report.

❖ **Finding the latest information on component certifications**

- 1 Point your Web browser to Availability and Certification Reports at <http://certification.sybase.com/>.
- 2 Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.
- 3 Select Search to display the availability and certification report for the selection.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

- 1 Point your Web browser to Technical Documents at <http://www.sybase.com/support/techdocs/>.
- 2 Click MySybase and create a MySybase profile.

**Sybase EBFs and  
software  
maintenance**

❖ **Finding the latest information on EBFs and software maintenance**

- 1 Point your Web browser to the Sybase Support Page at <http://www.sybase.com/support>.
- 2 Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
- 3 Select a product.
- 4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the “Technical Support Contact” role to your MySybase profile.

- 5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

## Conventions

**Table 1: Syntax conventions**

| Key             | Definition   |
|-----------------|--|
| command         | Command names, command option names, utility names, utility flags, and other keywords are in sans serif font.                                    |
| <i>variable</i> | Variables, or words that stand for values that you fill in, are in <i>italics</i> .  |
| { }             | Curly braces indicate that you choose at least one of the enclosed options. Do not include the braces in the command.                            |
| [ ]             | Brackets mean choosing one or more of the enclosed items is optional. Do not include the braces in the command.                                  |
| ( )             | Parentheses are to be typed as part of the command.  |
|                 | The vertical bar means you can select only one of the options shown.   |
| ,               | The comma means you can choose as many of the options shown as you like, separating your choices with commas to be typed as part of the command. |

## Accessibility features

This document is available in an HTML version that is specialized for accessibility. You can navigate the HTML with an adaptive technology such as a screen reader, or view it with a screen enlarger.

Open Client and Open Server documentation has been tested for compliance with U.S. government Section 508 Accessibility requirements. Documents that comply with Section 508 generally also meet non-U.S. accessibility guidelines, such as the World Wide Web Consortium (W3C) guidelines for Web sites.

---

**Note** You might need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

---

---

For information about how Sybase supports accessibility, see Sybase Accessibility at <http://www.sybase.com/accessibility>. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

**If you need help**

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the documentation or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

# Introducing DB-Library

This chapter gives an overview of DB-Library.

| Topic  | Page |
|--|------|
| Client/server architecture                     | 1    |
| The Open Client and Open Server products       | 3    |
| Data structures for communicating with servers | 6    |
| DB-Library/C programming                       | 6    |
| DB-Library/C routines                          | 12   |
| MIT Kerberos on DB-Library                     | 36   |
| Sample programs                                | 37   |

## Client/server architecture

Client/server architecture divides the work of computing between “clients” and “servers.”

Clients make requests of servers and process the results of those requests. For example, a client application might request data from a database server. Another client application might send a request to an environmental control server to lower the temperature in a room.

Servers respond to requests by returning data or other information to clients, or by taking some action. For example, a database server returns tabular data and information about that data to clients, and an electronic mail server directs incoming mail toward its final destination.

Client/server architecture has several advantages over traditional program architectures:

- Application size and complexity can be significantly reduced because common services are handled in a single location, a server. This simplifies client applications, reduces duplicate code, and makes application maintenance easier.

- Client/server architecture facilitates communication between varied applications. Client applications that use dissimilar communications protocols cannot communicate directly, but can communicate through a server that “speaks” both protocols.
- Client/server architecture allows applications to be developed with distinct components, which can be modified or replaced without affecting other parts of the application.

## **Types of clients**

A client is any application that makes requests of a server. Clients include:

- Stand-alone utilities provided with Adaptive Server Enterprise, such as isql and bcp
- Applications written using Open Client libraries
- Applications written using Open Client Embedded SQL™

## **Types of servers**

The Sybase product line includes servers and tools for building servers:

- Adaptive Server Enterprise is a database server. Adaptive Server Enterprise manages information stored in one or more databases.
- Open Server provides the tools and interfaces needed to create a custom server, also called an “Open Server application.”

An Open Server application can be any type of server. For example, an Open Server application can perform specialized calculations, provide access to real time data, or interface with services such as electronic mail. An Open Server application is created individually, using the building blocks provided by the Open Server Server-Library.

Adaptive Server Enterprise and Open Server applications are similar in some ways:

- Adaptive Server Enterprise and Open Server applications are both servers, responding to client requests.
- Clients communicate with both Adaptive Server Enterprise and Open Server applications through Open Client products.

But they also differ:

- An application programmer must create an Open Server application using Server-Library's building blocks and supplying custom code. Adaptive Server Enterprise is complete and does not require custom code.
- An Open Server application can be any kind of server, and can be written to understand any language. Adaptive Server Enterprise is a database server, and understands only Transact-SQL.
- An Open Server can communicate with "foreign" applications and servers that are not based on the TDS protocol, as well as Sybase applications and servers. Adaptive Server Enterprise can communicate directly only with Sybase applications and servers, although Adaptive Server Enterprise can communicate with foreign applications and servers by using an Open Server gateway application as an intermediary.

## The Open Client and Open Server products

Sybase provides two families of products to allow customers to write client and server application programs.: Open Client and Open Server.

### Open Client

Open Client provides customer applications, third-party products, and other Sybase products with the interfaces needed to communicate with Adaptive Server Enterprise and Open Server.

Open Client can be thought of as having two components: programming interfaces and network services.

The programming interfaces component of Open Client is made up of libraries designed for use in writing client applications: Client-Library, DB-Library, and CS-Library. (Both Open Client and Open Server include CS-Library, which contains utility routines that are useful to both client and server applications.

Open Client network services include Net-Library, which provides support for specific network protocols, such as TCP/IP.

## Open Server

Open Server provides the tools and interfaces needed to create custom server applications. Like Open Client, Open Server has a programming interfaces component and a network services component.

The programming interfaces component of Open Server contains Server-Library and CS-Library. (Both Open Client and Open Server include CS-Library, which contains utility routines that are useful to both client and server applications.)

Open Server network services are generally transparent.

## Open Client libraries

The libraries that make up Open Client are:

- DB-Library, a collection of routines for use in writing client applications. DB-Library includes a bulk copy library and the two-phase commit special library. DB-Library provides source-code compatibility for older Sybase applications.
- Client-Library, a collection of routines for use in writing client applications. Client-Library is a library designed to accommodate cursors and other advanced features.
- CS-Library, a collection of utility routines that are useful to both client and server applications. All Client-Library applications will include at least one call to CS-Library, because Client-Library routines use a structure which is allocated in CS-Library.

## What is in DB-Library/C?

---

**Note** DB-Library provides source code compatibility for older Sybase applications. Sybase encourages programmers to implement new applications with Client-Library or Embedded SQL.

---

DB-Library/C includes C routines and macros that allow an application to interact with Adaptive Server Enterprise and Open Server applications.



It includes routines that send commands to Adaptive Server Enterprise and Open Server applications and others that process the results of those commands. Other routines handle error conditions, perform data conversion, and provide a variety of information about the application's interaction with a server.

DB-Library/C also contains several header files that define structures and values used by the routines. Versions of DB-Library have been developed for a number of languages besides C, including COBOL, FORTRAN, Ada, and Pascal.

## **Comparing the library approach to Embedded SQL**

Either an Open Client library application or an Embedded SQL application can be used to send SQL commands to Adaptive Server Enterprise.

Generally, Embedded SQL is a superset of Transact-SQL. An Embedded SQL application includes Embedded SQL commands intermixed with the application's host language statements. The host language precompiler processes the Embedded SQL commands into calls to Client-Library routines and leaves the existing host-language statements as is. All version 10.0 or later precompilers use a runtime library composed solely of documented Client-Library and CS-Library calls.

In a sense, then, the precompiler transforms an Embedded SQL application into a Client-Library application.

An Open Client library application sends SQL commands through library routines, and does not require a precompiler.

Generally, an Embedded SQL application is easier to write and debug, but a library application can take fuller advantage of the flexibility and power of Open Client routines.

## **Data structures for communicating with servers**

A DB-Library/C application communicates with a server through one or more DBPROCESS structures. Through the DBPROCESS, commands are sent to the server and query results are returned to the application. One of the first routines an application typically calls is `dbopen`, which logs the application into the server and allocates and initializes a DBPROCESS. This DBPROCESS then serves as a connection between the application and the server. Most DB-Library/C routines require a DBPROCESS as the first parameter.

An application can have multiple open DBPROCESSes, connected to one or more servers. For instance, an application that has to perform database updates in the midst of processing the results of a query needs a separate DBPROCESS for each task. As another example, to select data from one server and update a database on another server, an application needs two DBPROCESSes—one for each server. Each DBPROCESS in an application functions independently of any other DBPROCESS.

The DBPROCESS structure points to a command buffer that contains language commands for transmission to the server. It also points to result rows returned from the server—either single rows or buffers of rows if buffering has been specified. In addition, it points to a message buffer that contains error and informational messages returned from the server.

The DBPROCESS also contains a wealth of information on various aspects of server interaction. Many of the DB-Library/C routines deal with extracting information from the DBPROCESS. Applications should access and manipulate components of the DBPROCESS structure only through DB-Library/C routines, and not directly.

One other important structure is the LOGINREC. It contains typical login information, such as the user name and password, which the `dbopen` routine uses when logging into a server. DB-Library/C routines can specify the information in the LOGINREC.

## **DB-Library/C programming**

An application programmer writes a DB-Library program, using calls to DB-Library routines to set up DB-Library structures, connect to servers, send commands, process results, and clean up. A DB-Library program is compiled and run in the same way as any other C language program.

Programming with DB-Library/C typically involves a few basic steps:

- 1 Logging into a server.
- 2 Placing language commands into a buffer and sending them to the server.
- 3 Processing the results, if any, returned from the server, one command at a time and one result row at a time. The results can be placed in program variables, where they can be manipulated by the application.
- 4 Handling DB-Library/C errors and server messages.
- 5 Closing the connection with the server.

The example below shows the basic framework of many DB-Library/C applications. The program opens a connection to a Adaptive Server Enterprise, sends a Transact-SQL select command to the server, and processes the set of rows resulting from the select. Note that this program does not include the error or message handling routines; those routines are illustrated in the sample programs included with DB-Library.

```
#include <sybfront.h>
#include <sybdb.h>
#include <syberror.h>

/* Forward declarations of the error handler and message
** handler.
*/
interr_handler();
intmsg_handler();

main()
{
    DBPROCESS      *dbproc;          /* The connection with */
                                      /* Adaptive Server Enterprise */
    LOGINREC       *login;           /* The login information */
    DBCHAR          name[40];
    DBCHAR          city[20];
    RETCODE         return_code;

    /* Initialize DB-Library */
    if (dbinit() == FAIL)
        exit(ERREXIT);

    /*
    ** Install user-supplied error-handling and message-
    ** handling routines. The code for these is omitted
    ** from this example for conciseness.
    */
}
```

```
*/
dberrhandle(err_handler);
dbmsghandle(msg_handler);

/* Get a LOGINREC */
login = dblogin();
DBSETLPWD(login, "server_password");
DBSETLAPP(login, "example");

/* Get a DBPROCESS structure for communication */
/* with Adaptive Server Enterprise. */
dbproc = dbopen(login, NULL);

/*
** Retrieve some columns from the "authors" table
** in the "pubs2" database.
*/

/* First, put the command into the command buffer. */
dbcmd(dbproc, "select au_lname, city from
               pubs2..authors");
dbcmd(dbproc, "
               where state = 'CA' ");

/*
** Send the command to Adaptive Server Enterprise and start
execution
*/
dbsqlxec(dbproc);

/* Process the command */
while ((return_code = dbresults(dbproc)) !=
       NO_MORE_RESULTS)
{
    if (return_code == SUCCEED)
    {
        /* Bind results to program variables. */
        dbbind(dbproc, 1, STRINGBIND, (DBINT)0, name);
        dbbind(dbproc, 2, STRINGBIND, (DBINT)0, city);

        /* Retrieve and print the result rows. */
        while (dbnextrow(dbproc) != NO_MORE_ROWS)
        {
            printf ("%s: %s\n", name, city);
        }
    }
}
```

```
    }  
  
    /* Close the connection to Adaptive Server Enterprise */  
    dbexit();  
}
```

The example illustrates features common to most DB-Library/C applications:

- Header files – Two header files, *sybfront.h* and *sybdb.h*, are required in all source files that contain calls to DB-Library/C routines. *sybfront.h* must appear first in the file. This file defines symbolic constants such as function return values, described in the reference pages in Chapter 2, “Routines” and the exit values STDEXIT and ERREXIT. These exit values can be used as the argument for the C standard library function `exit`. Since they are defined appropriately for the operating system running the program, their use provides a system-independent approach to exiting the program. *sybfront.h* also includes type definitions for datatypes that can be used in program variable declarations. These datatypes are described later.

*sybdb.h* contains additional definitions, most of which are meant to be used only by the DB-Library/C routines and should not be directly accessed by the program. Of chief importance in *sybdb.h* is the definition of the DBPROCESS structure. As discussed earlier, the DBPROCESS structure should be manipulated only through DB-Library/C routines; you should not access its components directly. To ensure compatibility with future releases of DB-Library/C, use the contents of *sybdb.h* only as documented in the reference pages in Chapter 2, “Routines.”

The third header file in the example, *syberror.h*, contains error severity values and should be included if the program refers to those values.

- `dbinit` – This routine initializes DB-Library/C. It must be the first DB-Library/C routine in the program. Not all DB-Library/C environments currently require the `dbinit` call. However, to ensure future compatibility and portability, you should include this call at the start of all DB-Library/C programs.
- `dberrhandle` and `dbmsghandle` – `dberrhandle` installs a user-supplied error-handling routine, which gets called automatically whenever the application encounters a DB-Library/C error. Similarly, `dbmsghandle` installs a message-handling routine, which gets called in response to informational or error messages returned from the server. The error and message handling routines are user-supplied. Sample handlers have not been supplied with this example, but are included with the sample programs provided with DB-Library. See the *Open Client and Open Server Programmers Supplement* for your platform.

- **dblogin** – This routine allocates a LOGINREC structure, which DB-Library/C will use to log in to the server. The two macros that follow set certain components of the LOGINREC. DBSETLUSER and DBSETLPWD set the user name and password that DB-Library/C will use when logging in. DBSETLAPP sets the name of the application, which will appear in Adaptive Server Enterprise's sysprocesses table. Routines are available for setting other aspects of the LOGINREC. However, in most environments these routines are optional; the LOGINREC contains default values for each of the values they set.
- **dbopen** – The dbopen routine opens a connection between the application and a server. It uses the LOGINREC supplied by dblogin to log in to the server. It returns a DBPROCESS structure, which serves as the conduit for information between the application and the server. After this routine has been called, the application is connected with Adaptive Server Enterprise and can now send Transact-SQL commands to Adaptive Server Enterprise and process any results.
- **dbcmd** – This routine fills the command buffer with Transact-SQL commands, which can then be sent to Adaptive Server Enterprise. Each succeeding call to dbcmd simply adds the supplied text to the end of any text already in the buffer. It is the programmer's responsibility to supply necessary blanks between words, such as the blank at the beginning of the text in the second dbcmd call in the example. Multiple commands can be included in the buffer. This example only shows how to send and process a single command, but DB-Library/C is designed to allow an application to send multiple commands to a server and process each command's set of results separately.
- **dbsqlxec** – This routine executes the command buffer; that is, it sends the contents of the buffer to Adaptive Server Enterprise, which parses and executes them.
- **dbresults** – This routine gets the results of the current Transact-SQL command ready for processing. In this case, the buffer contains a single command that returns rows, so the program is required to call dbresults one time. dbresults is called in a loop, however, because it is good programming practice to do so. It is recommended that dbresults always be called in a loop, as it is in this example, even when it is not strictly necessary.

- `dbbind` – `dbbind` binds result columns to program variables. In the example, the first call to `dbbind` binds the first result column to the program variable `city`. In other words, when the program reads a result row by calling `dbnextrow`, the contents of the first result column (`au_lname`) will get placed in the program variable `name`. The second `dbbind` call binds the second result column to the variable `city`.

The bind type of both bindings is `STRINGBIND`, one of several binding types available for character data. The binding type must correspond to the datatype of the specified program variable. In this example, the variable has a `DBCHAR` datatype, a DB-Library/C-defined datatype that accepts a `STRINGBIND` result. By means of the binding type parameter, `dbbind` supports a wide variety of type conversions, allowing the datatype of the receiving variable to differ from the datatype of the result column.

- `dbnextrow` – This routine reads a row and places the results in the program variables specified by the earlier `dbbind` calls. Each successive call to `dbnextrow` reads another result row, until the last row has been read and `NO_MORE_ROWS` is returned. Processing of the results must take place inside the `dbnextrow` loop, because each call to `dbnextrow` overwrites the earlier values in the program variables. This sample program merely prints each row's contents.
- `dbexit` – This routine closes the server connection and deallocates the `DBPROCESS`. It also cleans up any structures initialized by `dbinit`. It must be the last DB-Library/C routine in the program.

Although DB-Library/C contains a great number of routines, much can be accomplished with just the few routines shown in this example.

## DB-Library/C datatypes

DB-Library/C defines datatypes for Adaptive Server Enterprise data. These datatypes begin with “SYB” (for example, `SYBINT4`, `SYBCHAR`, `SYBMONEY`). Various routines require these datatypes as parameters. DB-Library/C and Server-Library/C also provide type definitions for use in program variable declarations. These types begin with the prefix “DB” (for example, `DBINT`, `DBCHAR`, `DBMONEY`, and so on) for DB-Library/C, and “SRV\_” for Server-Library/C (for example, `SRV_INT4`, `SRV_CHAR`, `SRV_MONEY`). By using them, you ensure that your program variables will be compatible.

See Types on page 412 for a list of Adaptive Server Enterprise datatypes and corresponding DB-Library/C program variable types. See the *Open Server Server-Library/C Reference Manual* for a list of Server-Library datatypes.

The `dbconvert_ps` routine provides a way to convert data from one server datatype to another. It supports conversion between most datatypes. Since Adaptive Server Enterprise and Open Server datatypes correspond directly to the DB-Library/C datatypes, you can use `dbconvert_ps` widely within your application. The routines that bind server result columns to program variables—`dbbind` and `dbaltbind`—also provide type conversion.

## DB-Library/C routines

The DB-Library/C routines and macros handle a large variety of tasks, which are divided in this section into a number of categories:

- Initialization
- Command processing
- Results processing
- Message and error handling
- Information retrieval
- Browse mode
- Text and image handling
- Datatype conversion
- Process control flow
- Remote procedure call processing
- Registered procedure call processing
- Gateway passthrough routines
- Datetime and money
- Cleanup
- Secure support
- Miscellaneous routines



The routines and macros are described in individual reference pages in Chapter 2, “Routines.” They all begin with the prefix “db.” The routines are named with lowercase letters; the macros are capitalized.

In addition, DB-Library/C includes two special libraries:

- Bulk Copy, described in Chapter 3, “Bulk Copy Routines”
- Two-Phase Commit Service, described in Chapter 4, “Two-Phase Commit Service”

The bulk copy routines begin with the prefix “bcp.” The two-phase commit routines have no standard prefix.

## Initialization

These routines set up and define the connection between the application program and a server. They handle such tasks as allocating and defining a LOGINREC structure, opening a connection to a server, and allocating a DBPROCESS structure. Only a few of the routines are absolutely necessary in every DB-Library/C program; in particular, an application requires dbinit, dblogin, and dbopen. The lists below specify the initialization routines in the approximate order in which a program is likely to call them.

### Initializing DB-Library/C

These are the top level routines that set up DB-Library’s internal environment:

- dbinit – initializes underlying structures used by DB-Library/C.
- dbsetversion – specifies a DB-Library version level.
- dbsetmaxprocs – sets the maximum number of simultaneously open DBPROCESS structures.
- dbgetmaxprocs – indicates the current maximum number of simultaneously open DBPROCESS structures.

### Setting up the LOGINREC

These routines place data in a LOGINREC. The LOGINREC contains the user information that DB-Library sends to the server when the program calls dbopen to open a connection.

- dblogin – allocates a LOGINREC structure for subsequent use by dbopen.

- DBSETLUSER – sets the server user name in the LOGINREC.
- DBSETLPWD – sets the server password in the LOGINREC.
- DBSETLAPP – sets the application name in the LOGINREC.
- DBSETLHOST – sets the host name in the LOGINREC.
- DBSETLCHARSET – sets the character set in the LOGINREC.
- DBSETLPACKET – sets the Tabular Data Stream™ (TDS) packet size for an application.
- dbgetpacket – returns the current TDS packet size.
- dbrpwset – adds a remote password to a LOGINREC structure. The server will use this password when it performs a remote procedure call on another server.
- dbrpwclr – clears all remote passwords from a LOGINREC structure.
- dbloginfree – frees a LOGINREC structure.

## Establishing a server connection

The application calls the following routines to set up and open a connection to a remote server:

- dbsetifile – specifies the interfaces file that dbopen will use to connect to a server.
- dbsetlogintime – sets the number of seconds DB-Library/C will wait for a server to respond to a request by dbopen for a DBPROCESS connection.
- dbopen – sets up communication with the network, logs into a server using the LOGINREC, initializes any options specified in the LOGINREC, and allocates a DBPROCESS. An application can open multiple connections to a server, each connection having its own DBPROCESS. An application can also open multiple connections to multiple servers.
- dbuse – sets the current database. This routine is equivalent to the Transact-SQL use command and can be called repeatedly in an application, any time when the connection is open.

## Command processing

An application can communicate with a server through language commands. For Adaptive Server Enterprise, the language is Transact-SQL. For Open Server, the language is whatever the Open Server has been programmed to understand. The application enters the commands into a command buffer, which the DBPROCESS points to. The application can place multiple commands in the command buffer, and the set of commands in the buffer is known as the command batch. The application then sends the command batch to the server, which executes the commands in the order entered in the buffer.

### Building the command batch

These routines add commands to the buffer or clear the buffer:

- `dbcmd` – adds text to the command buffer. It may be called repeatedly to add multiple commands, or parts of commands. The text added with each successive call is concatenated to the earlier text.
- `dbfcmd` – adds text to the command buffer using `sprintf`-type formatting. This routine is the same as `dbcmd`, except that it allows arguments to be substituted into the text.
- `dbfreebuf` – clears the command buffer. The command buffer is automatically cleared before a batch of commands is entered. To clear it at other times or when the `DBNOAUTOFREE` option has been set, use `dbfreebuf`.

### Accessing the command batch

These routines may be used to examine and copy parts of the command buffer:

- `dbgetchar` – returns a pointer to a particular character in the command buffer.
- `dbstrlen` – returns the length of the command buffer.
- `dbstrcpy` – copies a portion of the command buffer to a program variable. This routine is particularly valuable for debugging, because it can tell you exactly what was sent to the server.

### Executing the command batch

Once language commands have been entered in the buffer, they can be sent to a server for execution.

- `dbsqlsend` – sends the contents of the command buffer to a server for execution. Unlike `dbsqlxexec`, this routine does not wait for a response from the server. When `dbsqlsend` returns `SUCCEED`, `dbsqllok` must be called to verify the correctness of the command batch.
- `dbpoll` – when called between `dbsqlsend` (or `dbrpcsend`) and `dbsqllok`, checks if a server response has arrived for a `DBPROCESS`.
- `dbsqllok` – waits for results from the server and verifies the correctness of the instructions the server is responding to. This routine is used in conjunction with `dbsqlsend`, `dbrpcsend`, and `dbmoretext`. After a successful `dbsqllok` call, the application must call `dbresults` to process the results.
- `dbsqlxexec` – sends the contents of the command buffer to a server for execution. Once `dbsqlxexec` has returned `SUCCEED`, `dbresults` must be called to process the results. Calling `dbsqlxexec` is equivalent to calling `dbsqlsend` followed by `dbsqllok`.

## Setting and clearing command options

The application can set a number of Adaptive Server Enterprise and DB-Library/C command options. Among them are `DBPARSEONLY`, which causes Adaptive Server Enterprise to parse but not execute the command batch, and `DBBUFFER`, which provides buffering of result rows. For a list of all available options and their significance, see [Options on page 407](#).

- `dbsetopt` – sets an option
- `dbclopt` – clears an option
- `dbisopt` – determines whether a particular option is set

## Results processing

Once a command batch has been executed in the server, indicated by `dbsqlxexec` or `dbsqllok` returning `SUCCEED`, the application must process any results. Results can include:

- Success or failure indications from the server
- Result rows

Result rows are returned by `select` commands and `execute` commands on stored procedures that contain `select` commands.

There are two types of result rows: regular rows and compute rows. Regular rows are generated from columns in a `select` command's `select` list; compute rows are generated from columns in a `select` command's `compute` clause. Since these two types of rows contain very different data, the application must process them separately.

The results for each Transact-SQL command in a batch are returned to the application separately. Within each command's set of results, the result rows are processed one at a time.

If a command batch contains only a single Transact-SQL command and that command returns rows (for example, a `select` command), an application must call `dbresults` to process the results of the command.

If a command batch contains only a single Transact-SQL command and that command does not return rows (for example, a `use database` command or an `insert` command), an application does not have to call `dbresults` to process the results of the command. However, calling `dbresults` in these situations causes no harm. It may result in easier code maintenance if, after every command, you consistently call `dbresults` until it returns `NO_MORE_RESULTS`.

If the command batch contains more than one Transact-SQL command, an application must call `dbresults` once for every command in the batch, whether or not the command returns rows. For this reason, it is recommended that a DB-Library/C application always call `dbresults` in a loop after sending a command or commands to a server.

Table 1-1 lists Transact-SQL commands and the DB-Library/C functions required to process the results that they return:

**Table 1-1: DB-Library/C functions required to process Transact-SQL commands**

| <b>Transact-SQL command</b>                                   | <b>Required DB-Library/C functions</b>  |
|---|---|
| All Transact-SQL commands not listed elsewhere in this table. | dbresults. In some cases, for example dbcc, the command's normal output is considered by DB-Library/C to consist of errors and messages. The output is thus processed within a DB-Library/C application's error and message handlers instead of in the main program using dbnextrow or other DB-Library/C routines. |
| execute   | A DB-Library/C application must call dbresults once for every set of results that the stored procedure returns. In addition, if the stored procedure returns rows, the application must call dbnextrow or other DB-Library/C result-row routines.   |
| select  | dbresults. In addition, a DB-Library/C application must call dbnextrow or other DB-Library/C result-row routines.   |

## Setting up the results

dbresults sets up the results of the next command in the batch. dbresults must be called after dbsqlxexec or dbsqllok has returned SUCCEED, but before calls to dbbind or dbnextrow.

## Getting result data

The simplest way to get result data is to bind result columns to program variables, using dbbind and dbaltbind. Then, when the application calls dbnextrow to read a result row (see “Reading result rows” on page 19), DB-Library/C will automatically place copies of the columns' data into the program variables to which they are bound. The application must call dbbind and dbaltbind after a dbresults call but before the first call to dbnextrow.

You can also access a result column's data directly with dbdata and dbadata, which return pointers to the data. dbdata and dbadata have the advantage of providing access to the actual data, not a copy of the data. These routines are frequently used in conjunction with dbdatlen and dbadlen, which return the length of the data and are described in the section “Information retrieval” on page 24. When you are accessing data directly with these routines, you do not perform any preliminary binding of result columns to program variables. Simply call dbdata or dbadata after a dbnextrow call.

The following routines are used to retrieve result columns:

- `dbbind` – binds a regular row result column to a program variable.
- `dbbind_ps` – binds a regular row result column to a program variable, with precision and scale support for numeric and decimal variables.
- `dbaltbind` – binds a compute row result column to a program variable.
- `dbaltbind_ps` – binds a compute row result column to a program variable, with precision and scale support for numeric and decimal variables.
- `dbdata` – returns a pointer to the data for a regular row result column.
- `dbadata` – returns a pointer to the data for a compute row result column.
- `dbnullbind` – associates an indicator variable with a regular row result column.
- `dbanullbind` – associates an indicator variable with a compute-row column.
- `dbsetnull` – defines substitution values to be used when binding null values.
- `dbprtype` – converts a server type token into a readable string. Tokens are returned by various routines such as `dbcotype` and `dbaltop`.

## Reading result rows

Once `dbresults` has returned `SUCCEED` and any binding of columns to variables has been specified, the application is ready to process the results. The first step is to make the result rows available to the application. The `dbnextrow` routine accomplishes this. Each call to `dbnextrow` reads the next row returned from the server. The row is read directly from the network.

Once a row has been read in by `dbnextrow`, the application can perform any processing desired on the data in the row. If the result columns have been bound to program variables, the data in the row will have been automatically copied into the variables. Alternatively, the data is accessible through `dbdata` or `dbadata`.

Rows read in by `dbnextrow` may be automatically saved in a row buffer, if desired. The application accomplishes this by setting the `DBBUFFER` option with the `dbsetopt` routine. Row buffering is useful for applications that need to process result rows in a non-sequential manner. Without row buffering, the application must process each row as it is read in by `dbnextrow`, because the next call to `dbnextrow` will overwrite the row. If the application has allowed row buffering, the rows are added to a row buffer as they are read in by `dbnextrow`. The application can then use the `dbgetrow` routine to skip around in the buffer and return to previously read rows. Since row buffering carries a memory and performance penalty, use it with discretion. Note that row buffering has nothing to do with network buffering and is a completely independent issue.

Routines are also available to print result rows in a default format. Because the format is predetermined, these routines are of limited usefulness and are appropriate primarily for debugging.

Note that DB-Library/C processes results one command at a time. When the application has read all the results for one command, it must call `dbresults` again to set up the results for the next command in the command buffer. To ensure that all results are handled, Sybase strongly recommends that `dbresults` be called in a loop.

The following routines are used to process result rows:

- `dbnextrow` – reads in the next row. The return value from `dbnextrow` tells the application whether the row is a regular row or a compute row, whether the row buffer is full, and whether the last result row has been read.
- `DBCURROW` – returns the number of the row currently being read.
- `dbprhead` – prints default column headings for result rows. This routine is used in conjunction with `dbprrow`.
- `dbprrow` – prints all the result rows in a default format. When this routine is used, the program does not need to bind results or call `dbnextrow`.

## Canceling results

The following routines cancel results:

- `dbcancel` – cancels results from the current command batch. This routine cancels *all* the commands in the current batch.
- `dbcancquery` – cancels any rows pending from the most recently executed query.



As an example of the difference between these routines, consider an application that is processing the results of the language batch:

```
select * from pubs.titles
select * from pubs.authors
```

If the application calls `dbcanquery` while processing the titles rows, then the titles rows are discarded and the application must continue to call `dbresults` and process the rows from the next statement. If the application calls `dbcancel` while processing the titles rows, then DB-Library discards the titles rows and the results of all remaining, unprocessed commands in the batch. The application does not need to continue calling `dbresults` after calling `dbcancel`.

## Handling stored procedure results

A call to a stored procedure is made through either a remote procedure call, discussed in “Remote procedure call processing” on page 30, or a Transact-SQL `execute` command. The call can generate several types of results. First of all, a stored procedure that contains `select` statements will return result rows in the usual fashion. Each successive call to `dbresults` will access the set of rows from the next `select` statement in the stored procedure. These rows can be processed, as usual, with `dbnextrow`.

Second, stored procedures can contain “return parameters.” Return parameters, also called output parameters, provide stored procedures with a “call-by-reference” capability. Any change that a stored procedure makes internally to the value of an output parameter is available to the calling program. The calling program can retrieve output parameter values once it has processed all of the stored procedure’s result rows by calling `dbresults` and `dbnextrow`. A number of routines, described below, process return parameter values.

Third, stored procedures can return a status number.

To access a stored procedure’s output parameters and return status through the following routines:

- `dbnumrets` – returns the number of return parameter values generated by a stored procedure. If `dbnumrets` returns less than or equal to zero, no return parameter values are available.
- `dbretdata` – returns a pointer to a return parameter value.
- `dbretlen` – returns the length of a return parameter value.
- `dbrettype` – returns the datatype of a return parameter value.

- `dbretname` – returns the name of the return parameter associated with a particular value.
- `dbretstatus` – returns the stored procedure's status number.
- `dbhasretstat` – indicates whether the current command or remote procedure call generated a stored procedure status number. If `dbhasretstat` returns "FALSE," then no stored procedure status number is available.

## Setting results timeouts

By default, DB-Library will wait indefinitely for the results of a server command to arrive. Applications can use the routines below to specify a finite timeout period:

- `dbsettime` – sets the number of seconds that DB-Library/C will wait for a server response.
- `DBGETTIME` – gets the number of seconds that DB-Library/C will wait for a server response.

## Message and error handling

DB-Library/C applications must handle two types of messages and errors:

- Server messages and errors, which range in severity from informational messages to fatal errors. Server messages and errors are known to DB-Library/C applications as "messages." To list all possible Adaptive Server Enterprise messages, use the Transact-SQL command:

```
select * from sysmessages
```

For a list of Adaptive Server Enterprise messages, see the *Adaptive Server Enterprise System Administration Guide*. For a list of Open Server messages, see the *Open Server Server-Library/C Reference Manual*.

- DB-Library/C warnings and errors, known to DB-Library/C applications as "errors." For a list of DB-Library/C errors, see Errors on page 389.

Also, success or failure indications are returned by most DB-Library/C routines.

To handle server messages, DB-Library/C errors, and success or failure indications, a DB-Library/C application can:

- Test DB-Library/C routine return codes in the mainline code, handling failures on a case-by-case basis.
- Centralize message and error handling by installing a message handler and an error handler, which are then automatically called by DB-Library/C when a message or error occurs.

Sybase strongly recommends that all DB-Library/C applications use centralized message and error handling in addition to mainline error testing. Centralized message and error handling has substantial benefits for large or complex applications. For example:

- Centralized message and error handling reduces the need for mainline error-handling logic. This is because DB-Library/C calls an application's message and error handlers automatically whenever a message or error occurs.

Note, however, that even an application that uses centralized error and message handling will need some mainline error logic, depending on the nature of the application.

- Centralized message and error handling provides a mechanism for gracefully handling unexpected errors. An application using only mainline error-handling logic may not successfully trap errors which have not been anticipated.

To provide a DB-Library/C application with centralized message and error handling, the application programmer must write a message handler and an error handler and install them using `dbmsghandle` and `dberrhandle`.

The DB-Library/C routines for message and error handling are:

- `dbmsghandle` – installs a user function to handle server informational and error messages.
- `dberrhandle` – installs a user function to handle DB-Library/C error messages.
- `DBDEAD` – determines whether a particular `DBPROCESS` is dead. When a `DBPROCESS` is dead, the current DB-Library/C routine fails, causing the error handler to be called.

## Information retrieval

Information covering several areas, including regular result columns, compute result columns, row buffers, and the command state, can be retrieved from the DBPROCESS structure. As mentioned earlier, regular result columns correspond to columns in the select command's select list and compute result columns correspond to columns in the select command's optional compute clause.

### Regular result column information

These routines can be called after dbsqlxexec returns SUCCEED:

- `dbnumcols` – determines the number of columns in the current set of results.
- `dbcolname` – returns the name of a regular result column.
- `dbcollen` – returns the maximum length for a regular column's data.
- `dbcoltype` – returns the server datatype for a regular result column.
- `dbdatlen` – returns the actual length of a regular column's data. This routine is often used in conjunction with `dbdata`. The value returned by `dbdatlen` is different for each regular row read by `dbnextrow`.
- `dbvarylen` – indicates whether the column's data can vary in length.

### Compute result column information

These routines can be called after dbsqlxexec returns SUCCEED:

- `DBROWTYPE` – indicates whether the current result row is a regular row or a compute row.
- `dbnumcompute` – returns the number of compute clauses in the current set of results.
- `dbnumalts` – returns the number of columns in a compute row.
- `dbbylist` – returns the bylist for a compute row.
- `dbaltop` – returns the type of aggregate operator for a compute column.
- `dbalttype` – returns the datatype for a compute column.
- `dbaltlen` – returns the maximum length for a compute column's data.
- `dbaltcolid` – returns the column ID for a compute column.

- `dbadlen` – returns the actual length of a compute column’s data. This routine is often used in conjunction with `dbadata`. The value returned by `dbadlen` is different for each compute row read by `dbnextrow`.

## Row buffer information

These macros return information that can be useful when manipulating result rows in buffers:

- `DBFIRSTROW` – returns the number of the first row in the buffer.
- `DBLASTROW` – returns the number of the last row in the buffer.
- `dbgetrow` – reads the specified row in the row buffer. This routine provides the application with access to buffered rows that have been previously read by `dbnextrow`.
- `dbclrbuf` – drops rows from the row buffer.

## Command state information

These routines return information about the current state of the command batch. Several of them return information about the “current” command, that is, the command currently being processed by `dbresults`.

- `DBCURCMD` – returns the number of the current command in a batch.
- `dbgetoff` – checks for the existence of specified Transact-SQL constructs in the command buffer. This routine is used in conjunction with the `DBOFFSET` option.
- `DBMORECMDS` – indicates whether there are more commands in the batch.
- `DBCMDROW` – indicates whether the current command is one that can return rows (that is, a `select` or a stored procedure containing a `select`).
- `DBROWS` – indicates whether the current command actually did return rows.
- `DBCOUNT` – returns the number of rows affected by a command.
- `DBNUMORDERS` – returns the number of columns specified in a `select` command’s `order by` clause.
- `dbordercol` – returns the ID of a column appearing in a `select` command’s `order by` clause.

## Browse mode

Browse mode provides a means for browsing through database rows and updating their values a row at a time. From the standpoint of the program, the process involves several steps, because each row must be transferred from the database into program variables before it can be browsed and updated.

Since a row being browsed is not the actual row residing in the database, but is instead a copy residing in program variables, the program must be able to ensure that changes to the variables' values can be reliably used to update the original database row. In particular, in multiuser situations, the program needs to ensure that updates made to the database by one user do not unwittingly overwrite updates recently made by another user. This can be a problem because the application typically selects a number of rows from the database at one time, but the application's users browse and update the database one row at a time. A timestamp column in browsable database tables provides the information necessary to regulate this type of multiuser updating.

Browse mode routines also allow an application to handle ad hoc queries. Several routines return information that an application can use to examine the structure of a complicated ad hoc query to update the underlying database tables.

Conceptually, browse mode involves three steps:

- 1 Select result rows containing columns derived from one or more database tables.
- 2 Where appropriate, change values in columns of the result rows (*not* the actual database rows), one row at a time.
- 3 Update the original database tables, one row at a time, using the new values in the result rows.

These steps are implemented in a program as follows:

- 1 Execute a `select` command, generating result rows containing result columns. The `select` command must include the `for browse` option.
- 2 Copy the result column values into program variables, one row at a time.
- 3 If appropriate, change the values of the variables (possibly in response to user input).
- 4 If appropriate, execute an `update` command that updates the database row corresponding to the current result row. To handle multiuser updates, the `where` clause of the `update` command must reference the timestamp column. Such a `where` clause can be obtained through the `dbqual` function.

5 Repeat steps 2, 3, and 4 for each result row.

To use browse mode, the following conditions must be true:

- The select command must end with the key words for browse.
- The table(s) to be updated must be “browsable” (that is, each must have a unique index and a timestamp column). Note that because a browse mode table has unique rows, the keyword `distinct` has no effect in a select against a browse-mode table.
- The result columns to be used in the updates must be “updatable”—they must be derived from browsable tables and cannot be the result of SQL expressions, such as `max(colname)`. In other words, there must be a valid correspondence between the result column and the database column to be updated. In addition, browse mode usually requires two connections (DBPROCESS pointers)—one for selecting the data and another for performing updates based on the selected data.

For examples of browse-mode programming, see the sample programs, *example6.c* and *example7.c*, included with DB-Library. See “Sample programs” on page 37.

The following constitute the browse-mode routines:

- `dbqual` – returns a pointer to a where clause suitable for use in updating the current row in a browsable table.
- `dbfreequal` – frees the memory allocated by `dbqual`.
- `dbtsnewval` – returns the new value of the timestamp column after a browse-mode update.
- `dbtsnewlen` – returns the length of the new value of the timestamp column after a browse-mode update.
- `dbtsput` – puts the new value of the timestamp column into the given table’s current row in the DBPROCESS.
- `dbcoldbrowse` – indicates whether the source of a result column is updatable through browse mode.
- `dbcoldsource` – returns a pointer to the name of the database column from which the specified result column was derived.
- `dbtabbrowse` – indicates whether a particular table is updatable using browse mode.
- `dbtabcount` – returns the number of tables involved in the current select command.

- `dbtabname` – returns the name of a table based on its number.
- `dbtabsource` – returns the name and number of the table from which a particular result column was derived.

## Text and image handling

The text and image Adaptive Server Enterprise datatypes are designed to hold large text or image values. The text datatype will hold up to 2,147,483,647 bytes of printable characters; the image datatype will hold up to 2,147,483,647 bytes of binary data.

Because they can be so large, text and image values are not actually stored in database tables. Instead, a pointer to the text or image value is stored in the table. This pointer is called a “text pointer.”

To ensure that competing applications do not wipe out one another’s modifications to the database, a timestamp is associated with each text or image column. This timestamp is called a “text timestamp.”

A DB-Library/C application that uses `dbwritetext` to insert text or image data into a table must perform the following steps:

- 1 Use the insert command to insert all data into the row except the text or image value.
- 2 Use the update command to update the row, setting the value of the text or image column to NULL. This step is necessary because a text or image column row that contains a null value will have a valid text pointer only if the null value was explicitly entered with the update statement.
- 3 Use the select command to select the row. You must specifically select the column that is to contain the text or image value. This step is necessary to provide the application’s DBPROCESS with correct text pointer and text timestamp information. The application should throw away the data returned by this select.
- 4 Call `dbtxtptr` to retrieve the text pointer from the DBPROCESS.
- 5 Call `dbtxtimestamp` to retrieve the text timestamp from the DBPROCESS.
- 6 Write the text or image value to Adaptive Server Enterprise. An application can either:
  - Write the value with a single call to `dbwritetext`, or
  - Write the value in chunks, using `dbwritetext` and `dbmoretext`.



- 7 If the application plans to make another update to this text or image value, it may want to save the new text timestamp that is returned by Adaptive Server Enterprise at the conclusion of a successful `dbwritetext` operation. The new text timestamp may be accessed using `dbtxtsnewval` and stored for later retrieval using `dbtxtsput`.

Several routines are available to facilitate the process of updating text and image columns in database tables:

- `dbreadtext` – reads a text or an image value from Adaptive Server Enterprise.
- `dbwritetext` – sends a text or an image value to Adaptive Server Enterprise.
- `dbmoretext` – sends part of a text or an image value to Adaptive Server Enterprise.
- `dbtxptr` – returns the text pointer for a column in the current results row.
- `dbtxtimestamp` – returns the value of the text timestamp for a column in the current results row.
- `dbtxtsnewval` – returns the new value of a text timestamp after a call to `dbwritetext`.
- `dbtxtsput` – puts the new value of a text timestamp into the specified column of the current row in the DBPROCESS.

## Datatype conversion

DB-Library/C supports conversions between most server datatypes with the `dbconvert` and `dbconvert_ps` routines. For information on server datatypes, see *Types* on page 412.

The `dbbind`, `dbbind_ps`, `dbaltbind`, and `dbaltbind_ps` routines, which bind result columns to program variables, can also be used to perform type conversion. Each of these routines contain a parameter that specifies the datatype of the receiving program variable. If the data being returned from the server is of a different datatype, DB-Library/C will usually convert it automatically to the type specified by the parameter.

These routines are used to perform datatype conversion:

- `dbconvert_ps` – converts data from one server datatype to another, with precision and scale support for numeric and decimal datatypes.
- `dbconvert` – converts data from one server datatype to another.

- `dbwillconvert` – indicates whether a specified datatype conversion is supported.

## Process control flow

These routines allow the application to schedule its actions around its interaction with a server:

- `dbsetbusy` – calls a user-supplied function when DB-Library/C is reading or waiting to read results from the server.
- `dbsetidle` – calls a user-supplied function when DB-Library/C is finished reading from the server.
- `dbsetinterrupt` – calls user-supplied functions to handle interrupts while waiting on a read from the server.
- `DBIORDESC` (UNIX only) – provides access to the UNIX file descriptor used to read data coming from the server, allowing the application to respond to multiple input data streams.
- `DBIOWDESC` (UNIX only) – provides access to the UNIX file descriptor used to write data to the server, allowing the application to effectively utilize multiple input and output data streams.
- `DBRBUF` (UNIX only) – determines whether the DB-Library/C network buffer contains any unread bytes.

## Remote procedure call processing

A remote procedure call is simply a call to a stored procedure residing on a remote server. Either an application or another server makes the call. A remote procedure call made by an application has the same effect as an `execute` command: It executes the stored procedure, generating results accessible through `dbresults`. However, a remote procedure call is often more efficient than an `execute` command. Note that if the procedure being executed resides on a server other than the one to which the application is directly connected, commands executed within the procedure cannot be rolled back.

A server can make a remote procedure call to another server. This occurs when a stored procedure being executed on one server contains an `execute` command for a stored procedure on another server. The `execute` command causes the first server to log in to the second server and perform a remote procedure call on the procedure. This happens without any intervention from the application, although the application can specify the remote password that the first server uses to log in.

The following routines are used to perform remote procedure calls:

- `dbrpcinit` – initializes a remote procedure call to a stored procedure.
- `dbrpcparam` – adds a parameter to a remote procedure call.
- `dbrpcsend` – signals the end of a remote procedure call, causing the server to begin executing the specified procedure.
- `dbpoll` – when called between `dbsqlsend` (or `dbrpcsend`) and `dbsqllok`, checks if a server response has arrived for a `DBPROCESS`.
- `dbsqllok` – waits for results from the server and verifies the correctness of the instructions the server is responding to. This routine is used in conjunction with `dbsqlsend`, `dbrpcsend`, and `dbmoretext`. After a successful `dbsqllok` call, the application must call `dbresults` to process the results.

## Registered procedure call processing

A registered procedure is a procedure that is defined and installed in a running Open Server. Registered procedures require Open Server version 2.0 or later. At this time, registered procedures are not supported by Adaptive Server Enterprise.

For DB-Library/C applications, registered procedures provide a way for inter-application communication and synchronization. This is because DB-Library/C applications connected to an Open Server can “watch” for a registered procedure to execute. When the registered procedure executes, applications watching for it receive a notification that includes the procedure’s name and the arguments it was called with.

---

**Note** DB-Library/C applications may create only a special type of registered procedure, known as a “notification procedure.” A notification procedure differs from a normal Open Server registered procedure in that it contains no executable statements.

---

For example, suppose the following:

- `stockprice` is a real-time DB-Library/C application monitoring stock prices.
- `price_change` is a notification procedure created in Open Server by the `stockprice` application. `price_change` takes as parameters a stock name and a price differential.
- `sellstock`, an application that puts stock up for sale, has requested to be notified when `price_change` executes.

When `stockprice`, the monitoring application, becomes aware that the price of Extravagant Auto Parts stock has risen \$1.10, it executes `price_change` with the parameters “Extravagant Auto Parts” and “+1.10”.

When `price_change` executes, Open Server sends `sellstock` a notification containing the name of the procedure (`price_change`) and the arguments passed to it (“Extravagant Auto Parts” and “+1.10”). `sellstock` uses the information contained in the notification to decide to put 100 shares of Extravagant Auto Parts stock up for sale.

`price_change` is the means through which the `stockprice` and `sellstock` applications communicate.

Registered procedures as a means of communication have the following advantages:

- A single call to execute a registered procedure can result in many client applications being notified that the procedure has executed. The application executing the procedure does not need to know how many, or which, clients have requested notifications.
- The registered procedure communication mechanism is server-based. Open Server acts as a central repository for connection addresses. Because of this, client applications can communicate without having to connect directly to each other. Instead, each client simply connects to the server.

A DB-Library/C application can:

- Create a registered procedure in Open Server
- Drop a registered procedure
- List all registered procedures defined in Open Server
- Request to be notified when a particular registered procedure is executed
- Drop a request to be notified when a particular registered procedure is executed

- List all registered procedure notifications
- Execute a registered procedure
- Install a user-supplied handler to be called when an application receives notification that a registered procedure has executed
- Poll Open Server to see if any registered procedure notifications are pending

The following are registered procedure routines:

- `dbnpcreate` – creates a notification procedure.
- `dbnpdefine` – defines a notification procedure.
- `dbregdrop` – drops a registered procedure.
- `dbreglist` – returns a list of all registered procedures currently defined in Open Server.
- `dbreghandle` – installs a handler routine for a registered procedure notification.
- `dbreginit` – initiates execution of a registered procedure.
- `dbregnowatch` – cancels a request to be notified when a registered procedure executes.
- `dbregparam` – defines a parameter for a registered procedure.
- `dbregexec` – executes a registered procedure.
- `dbregwatch` – requests to be notified when a registered procedure executes.
- `dbregwatchlist` – returns a list of registered procedures that a DBPROCESS is watching for.
- `dbpoll` – in an application that uses registered procedure notifications, this routine is used to check whether any notifications have arrived.

## Gateway passthrough routines

Passthrough routines can be called in Open Server gateway applications. They allow a DB-Library/C application to send and receive whole Tabular Data Stream™ (TDS) packets and set TDS packet size.

TDS is an application protocol used for the transfer of requests and request results between clients and servers. These routines are used with the `srvrecvpassthru` and `srvsendpassthru` Open Server Server-Library routines:

- `dbrecvpass thru` – receives a TDS packet from Open Server.
- `dbsendpass thru` – sends a TDS packet to Open Server.

See the *Open Server Server-Library/C Reference Manual* for descriptions of `srvrecvpass thru` and `srvsendpass thru`.

## Datetime and money

These routines manipulate datetime and money datatypes. datetime and money datatypes come in long versions, `DBDATETIME` and `DBMONEY`, and short (4-byte) versions, `DBDATETIME4` and `DBMONEY4`. All of the `DBDATETIME4` routines listed below are also available for `DBDATETIME`, and all `DBMONEY4` routines are available for `DBMONEY`. For example, `dbmny4add`, listed below, is also available as `dbmnyadd`.

- `dbdate4cmp` – compares two `DATETIME4` values.
- `dbdate4zero` – initializes a `DBDATETIME4` value.
- `dbmny4add` – adds two `DBMONEY4` values.
- `dbmny4cmp` – compares two `DBMONEY4` values.
- `dbmny4copy` – copies a `DBMONEY4` value.
- `dbmny4divide` – divides one `DBMONEY4` value by another.
- `dbmny4minus` – negates a `DBMONEY4` value.
- `dbmny4mul` – multiplies a `DBMONEY4` value.
- `dbmny4sub` – subtracts a `DBMONEY4` value.
- `dbmny4zero` – initializes a `DBMONEY4` value.
- `dbmnydec` – decrements a `DBMONEY` value.
- `dbmnydown` – divides a `DBMONEY` value by a positive integer.
- `dbmnyinc` – increments a `DBMONEY` value.
- `dbmnyinit` – prepares a `DBMONEY` value for calls to `dbmnyndigit`.
- `dbmnymaxneg` – returns the maximum negative `DBMONEY` value.
- `dbmnymaxpos` – returns the maximum positive `DBMONEY` value.
- `dbmnyndigit` – returns the rightmost digit of a `DBMONEY` value as a `DBCHAR`.

- `dbmynscale` – multiplies a `DBMONEY` value and adds a specified amount.

## Cleanup

These routines sever the connection between the application and a server:

- `dbexit` – closes and deallocates all `DBPROCESS` structures. This routine also cleans up any structures initialized by `dbinit`.
- `dbcclose` – closes and deallocates a single `DBPROCESS` structure.

## Secure support

These routines provide security for DB-Library applications running against Adaptive Server Enterprise:

- `DBSETLENCRYPT` – specifies whether or not password encryption is to be used when logging into Adaptive Server Enterprise.
- `dbsechandle` – installs user functions to handle secure logins.
- `bcp_options` – sets bulk copy options, including `BCPLABELED`, the security label option.

---

**Note** Calling `DBSETLENCRYPT` causes an error unless you first set the DB-Library version to 10.0. Use `dbsetversion` to set the DB-Library version to 10.0 before calling `DBSETLENCRYPT`.

---

## Miscellaneous routines

These routines may be useful in some applications:

- `dbsetavail` – marks a `DBPROCESS` as being available for general use.
- `DBISAVAIL` – indicates whether a `DBPROCESS` is available for general use.
- `dbname` – returns the name of the current database.
- `dbchange` – indicates whether a command batch has changed the current database.

- `dbsetuserdata` – uses a `DBPROCESS` structure to save a pointer to user-allocated data. This routine, along with `dbgetuserdata`, allows the application to associate user data with a particular `DBPROCESS`. One important use for these routines is to transfer information between a server message handler and the program code that triggered it.
- `dbgetuserdata` – returns a pointer to user-allocated data from a `DBPROCESS` structure.
- `dbreadpage` – reads in a page of binary data from Adaptive Server Enterprise.
- `dbwritepage` – writes a page of binary data to Adaptive Server Enterprise.
- `dbsetconnect` – sets server connection information in this routine.

## Two-phase commit service special library

The routines in this library allow an application to coordinate updates among two or more Adaptive Server Enterprises.

See Chapter 4, “Two-Phase Commit Service.”

## MIT Kerberos on DB-Library

DB-Library uses the MIT Kerberos security mechanism to provide network and mutual authentication services. This feature allows older Sybase applications to use Kerberos authentication services, with less need for modification and recompilation.

These DB-Library macros enable Kerberos support:

- `DBSETLNETWORKAUTH` – enables or disables network base authentication.
- `DBSETLMUTUALAUTH` – enables or disables mutual authentication of the connection’s security mechanism.



- DBSETLSERVERPRINCIPAL – sets the server’s principal name, if required.

---

**Note** DB-Library only supports network authentication and mutual authentication services in the Kerberos security mechanism.

---

#### ❖ Installing MIT-Kerberos on DB-Library

These steps provide basic information on installing MIT Kerberos on DB-Library. For more detailed information, refer to *Installation and Release Bulletin* for Sybase SDK DB-Lib Kerberos Authentication Option 15.5.

- 1 Purchase Sybase SDK DB-Lib Kerberos Authentication Option 15.5.
- 2 Install Sybase SDK DB-Lib Kerberos Authentication Option 15.5 over SDK 15.5.
- 3 In DB-Library, include *sybdbn.h* instead of *sybdb.h*.
- 4 Using *dbsetversion*, set the DB-Library version to *DBVERSION\_100* or above.
- 5 Call one or more of the following APIs:  
    DBSETLNETWORKAUTH(LOGINREC \*loginrec, DBBOOL enable)  
    DBSETLMUTUALAUTH(LOGINREC \*loginrec, DBBOOL enable)  
    DBSETLSERVERPRINCIPAL(LOGINREC \*loginrec, char \*name)
- 6 Recompile DB-Library.

## Sample programs

Several sample programs are provided that demonstrate the use of DB-library routines and their functionality. These samples are available in the following directory:

- `$SYBASE/$SYBASE_OCS/sample/dblibrary` on UNIX
- `%SYBASE%\%SYBASE_OCS%\sample\dblib` on Windows

See the *Open Client and Open Server Programmers Supplement* for your platform.



This chapter contains a reference page for each DB-Library routine.

| <b>Routines</b> | <b>Description</b>   | <b>Page</b> |
|-----------------|--|-------------|
| db12hour        | Determines whether the specified language uses 12-hour or 24-hour time.  | 48          |
| dbadata         | Returns a pointer to the data for a compute column.  | 49          |
| dbadlen         | Returns the actual length of the data for a compute column.  | 52          |
| dbaltbind       | Binds a compute column to a program variable.  | 54          |
| dbaltbind_ps    | Binds a compute column to a program variable, with precision and scale support for numeric and decimal datatypes.        | 59          |
| dbaltcolid      | Returns the column ID for a compute column.  | 65          |
| dbaltlen        | Returns the maximum length of the data for a particular compute column.  | 66          |
| dbaltop         | Returns the type of aggregate operator for a particular compute column.  | 67          |
| dbalttype       | Returns the datatype for a compute column.   | 68          |
| dbaltutype      | Returns the user-defined datatype for a compute column.  | 69          |
| dbanullbind     | Associates an indicator variable with a compute-row column.  | 70          |
| dbbind          | Binds a regular result column to a program variable.   | 72          |
| dbbind_ps       | Binds a regular result column to a program variable, with precision and scale support for numeric and decimal datatypes. | 77          |
| dbbufsize       | Returns the size of a DBPROCESS row buffer.  | 82          |
| dbbylist        | Returns the bylist for a compute row.  | 83          |
| dbcancel        | Cancels the current command batch.   | 84          |
| dbcquery        | Cancels any rows pending from the most recently executed query.  | 85          |
| dbchange        | Determines whether a command batch has changed the current database.   | 86          |
| dbcharsetconv   | Indicates whether the server is performing character set translation.  | 87          |
| dbclose         | Closes and deallocate a single DBPROCESS structure.  | 88          |
| dbclrbuf        | Drops rows from the row buffer.  | 88          |

---

| <b>Routines</b> | <b>Description</b>   | <b>Page</b> |
|-----------------|--|-------------|
| dbclopt         | Clears an option set by dbsetopt.  | 89          |
| dbcmd           | Adds text to the DBPROCESS command buffer.   | 91          |
| DBCMDROW        | Determines whether the current command can return rows.  | 92          |
| dbccolbrowse    | Determines whether the source of a regular result column is updatable using the DB-Library browse-mode facilities.                                     | 93          |
| dbcollen        | Returns the maximum length of the data in a regular result column.   | 94          |
| dbccolname      | Returns the name of a regular result column.   | 95          |
| dbccolsource    | Returns a pointer to the name of the database column from which the specified regular result column was derived.                                       | 97          |
| dbccoltype      | Returns the datatype for a regular result column.  | 98          |
| dbccoltypeinfo  | Returns precision and scale information for a regular result column of type numeric or decimal.  | 99          |
| dbcolutype      | Returns the user-defined datatype for a regular result column.   | 100         |
| dbconvert       | Converts data from one datatype to another.  | 102         |
| dbconvert_ps    | Converts data from one datatype to another, with precision and scale support for numeric and decimal datatypes.  | 106         |
| DBCOUNT         | Returns the number of rows affected by a Transact-SQL command.   | 112         |
| DBCURCMD        | Returns the number of the current command.   | 113         |
| DBCURROW        | Returns the number of the row currently being read.  | 114         |
| dbcursor        | Inserts, updates, deletes, locks, or refreshes a particular row in the fetch buffer.   | 115         |
| dbcursorbind    | Registers the binding information on the cursor columns.   | 117         |
| dbcursorclose   | Closes the cursor associated with the given handle and release all the data belonging to it.   | 119         |
| dbcursorcolinfo | Returns column information for the specified column number in the open cursor.   | 120         |
| dbcursorfetch   | Fetches a block of rows into the program variables declared by the user in dbcursorbind.   | 121         |
| dbcursorinfo    | Returns the number of columns and the number of rows in the keyset if the keyset hit the end of the result set.  | 123         |
| dbcursoropen    | Opens a cursor and specify the scroll option, concurrency option, and the size of the fetch buffer (the number of rows retrieved with a single fetch). | 124         |
| dbdata          | Returns a pointer to the data in a regular result column.  | 128         |
| dbdate4cmp      | Compares two DBDATETIME4 values.   | 129         |

| <b>Routines</b> | <b>Description</b>  | <b>Page</b> |
|-----------------|---|-------------|
| dbdate4zero     | Initializes a DBDATETIME4 variable to Jan 1, 1900 12:00AM.  | 130         |
| dbdatechar      | Converts an integer component of a DBDATETIME value into character format.  | 131         |
| dbdatecmp       | Compares two DBDATETIME values.   | 132         |
| dbdatecrack     | Converts a machine-readable DBDATETIME value into user-accessible format.   | 133         |
| dbdatename      | Converts the specified component of a DBDATETIME structure into its corresponding character string.                                   | 135         |
| dbdateorder     | Returns the date component order for a given language.  | 138         |
| dbdatepart      | Returns the specified part of a DBDATETIME value as a numeric value.  | 139         |
| dbdatezero      | Initializes a DBDATETIME value to Jan 1, 1900 12:00:00:000AM.   | 140         |
| dbdatlen        | Returns the length of the data in a regular result column.  | 141         |
| dbdayname       | Determines the name of a specified weekday in a specified language.   | 142         |
| DBDEAD          | Determines whether a particular DBPROCESS is dead.  | 143         |
| dberrhandle     | Installs a user function to handle DB-Library errors.   | 144         |
| dbexit          | Closes and deallocate all DBPROCESS structures, and clean up any structures initialized by dbinit.                                    | 148         |
| dbfcmd          | Adds text to the DBPROCESS command buffer using C runtime library sprintf-type formatting.  | 149         |
| DBFIRSTROW      | Returns the number of the first row in the row buffer.  | 152         |
| dbfree_xlate    | Frees a pair of character set translation tables.   | 153         |
| dbfreebuf       | Clears the command buffer.  | 154         |
| dbfreeequal     | Frees the memory allocated by dbqual.   | 155         |
| dbfreesort      | Frees a sort order structure allocated by dbloadsrt.  | 155         |
| dbgetchar       | Returns a pointer to a character in the command buffer.   | 157         |
| dbgetcharset    | Gets the name of the client character set from the DBPROCESS structure.   | 157         |
| dbgetloginfo    | Transfers Tabular Data Stream (TDS) login response information from a DBPROCESS structure to a newly allocated DBLOGININFO structure. | 158         |
| dbgetusername   | Returns the user name from a LOGINREC structure.  | 160         |
| dbgetmaxprocs   | Determines the current maximum number of simultaneously open DBPROCESSes.   | 161         |
| dbgetnatlang    | Gets the national language from the DBPROCESS structure.  | 162         |

---

| <b>Routines</b> | <b>Description</b>   | <b>Page</b> |
|-----------------|--|-------------|
| dbgetoff        | Checks for the existence of Transact-SQL constructs in the command buffer.   | 162         |
| dbgetpacket     | Returns the TDS packet size currently in use.  | 164         |
| dbgetrow        | Reads the specified row in the row buffer.   | 165         |
| DBGETTIME       | Returns the number of seconds that DB-Library will wait for a server response to a SQL command.                          | 167         |
| dbgetuserdata   | Returns a pointer to user-allocated data from a DBPROCESS structure.   | 167         |
| dbhasretstat    | Determines whether the current command or remote procedure call generated a return status number.                        | 168         |
| dbinit          | Initialize DB-Library.   | 170         |
| DBIORDESC       | (UNIX only) Provides program access to the UNIX file descriptor used by a DBPROCESS to read data coming from the server. | 170         |
| DBIOWDESC       | (UNIX only) Provides program access to the UNIX file descriptor used by a DBPROCESS to write data to the server.         | 172         |
| DBISAVAIL       | Determines whether a DBPROCESS is available for general use.   | 173         |
| dbisopt         | Checks the status of a server or DB-Library option.  | 173         |
| DBLASTROW       | Returns the number of the last row in the row buffer.  | 174         |
| dbload_xlate    | Loads a pair of character set translation tables.  | 175         |
| dbloadsort      | Loads a server sort order.   | 176         |
| dblogin         | Allocates a login record for use in dbopen.  | 177         |
| dbloginfree     | Frees a login record.  | 179         |
| dbmny4add       | Adds two DBMONEY4 values.  | 179         |
| dbmny4cmp       | Compares two DBMONEY4 values.  | 180         |
| dbmny4copy      | Copies a DBMONEY4 value.   | 181         |
| dbmny4divide    | Divides one DBMONEY4 value by another.   | 182         |
| dbmny4minus     | Negates a DBMONEY4 value.  | 183         |
| dbmny4mul       | Multiplies two DBMONEY4 values.  | 184         |
| dbmny4sub       | Subtracts one DBMONEY4 value from another.   | 185         |
| dbmny4zero      | Initializes a DBMONEY4 variable to \$0.0000.   | 186         |
| dbmnyadd        | Adds two DBMONEY values.   | 187         |
| dbmnycmp        | Compares two DBMONEY values.   | 188         |
| dbmnycopy       | Copies a DBMONEY value.  | 189         |
| dbmnydec        | Decrements a DBMONEY value by one ten-thousandth of a dollar.  | 190         |

| <b>Routines</b> | <b>Description</b>  | <b>Page</b> |
|-----------------|---|-------------|
| dbmnydivide     | Divides one DBMONEY value by another.   | 191         |
| dbmnydown       | Divides a DBMONEY value by a positive integer.  | 192         |
| dbmnyinc        | Increments a DBMONEY value by one ten-thousandth of a dollar.   | 194         |
| dbmnyinit       | Prepares a DBMONEY value for calls to dbmnyndigit.  | 194         |
| dbmnymaxneg     | Returns the maximum negative DBMONEY value supported.   | 196         |
| dbmnymaxpos     | Returns the maximum positive DBMONEY value supported.   | 197         |
| dbmnyminus      | Negates a DBMONEY value.  | 198         |
| dbmnymul        | Multiplies two DBMONEY values.  | 199         |
| dbmnyndigit     | Returns the rightmost digit of a DBMONEY value as a DBCHAR.   | 200         |
| dbmnyscale      | Multiplies a DBMONEY value by a positive integer and add a specified amount.                                | 206         |
| dbmnysub        | Subtracts one DBMONEY value from another.   | 208         |
| dbmnyzero       | Initializes a DBMONEY value to \$0.0000.  | 209         |
| dbmonthname     | Determines the name of a specified month in a specified language.   | 209         |
| DBMORECMDS      | Indicates whether there are more commands to be processed.  | 210         |
| dbmoretext      | Sends part of a text or image value to the server.  | 211         |
| dbmsghandle     | Installs a user function to handle server messages.   | 212         |
| dbname          | Returns the name of the current database.   | 216         |
| dbnextrow       | Reads the next result row into the row buffer and into any program variables that are bound to column data. | 217         |
| dbnpcreate      | Creates a notification procedure.   | 219         |
| dbnpdefine      | Defines a notification procedure.   | 222         |
| dbnullbind      | Associates an indicator variable with a regular result row column.  | 224         |
| dbnumalts       | Returns the number of columns in a compute row.   | 225         |
| dbnumcols       | Determines the number of regular columns for the current set of results.                                    | 225         |
| dbnumcompute    | Returns the number of compute clauses in the current set of results.  | 227         |
| DBNUMORDERS     | Returns the number of columns specified in a Transact-SQL select statement's order by clause.               | 227         |
| dbnumrets       | Determines the number of return parameter values generated by a stored procedure.                           | 228         |

---

| <b>Routines</b> | <b>Description</b>   | <b>Page</b> |
|-----------------|--|-------------|
| dbopen          | Creates and initialize a DBPROCESS structure.  | 229         |
| dbordercol      | Returns the ID of a column appearing in the most recently executed query's order by clause.                | 233         |
| dbpoll          | Checks if a server response has arrived for a DBPROCESS.   | 234         |
| dbprhead        | Prints the column headings for rows returned from the server.  | 239         |
| dbprrow         | Prints all the rows returned from the server.  | 240         |
| dbprtype        | Converts a token value to a readable string.   | 241         |
| dbqual          | Returns a pointer to a where clause suitable for use in updating the current row in a browsable table.     | 242         |
| DBRBUF          | (UNIX only) Determines whether the DB-Library network buffer contains any unread bytes.                    | 246         |
| dbreadpage      | Reads a page of binary data from the server.   | 247         |
| dbreadtext      | Reads part of a text or image value from the server.   | 248         |
| dbrectos        | Records all SQL commands sent from the application to the server.  | 250         |
| dbrecvpassthru  | Receives a TDS packet from a server.   | 251         |
| dbregdrop       | Drops a registered procedure.  | 253         |
| dbregexec       | Executes a registered procedure.   | 254         |
| dbreghandle     | Installs a handler routine for a registered procedure notification.  | 256         |
| dbreginit       | Initiates execution of a registered procedure.   | 260         |
| dbreglist       | Returns a list of registered procedures currently defined in Open Server.                                  | 262         |
| dbregnowatch    | Cancels a request to be notified when a registered procedure executes.                                     | 263         |
| dbregparam      | Defines or describes a registered procedure parameter.   | 265         |
| dbregwatch      | Requests to be notified when a registered procedure executes.  | 269         |
| dbregwatchlist  | Returns a list of registered procedures that a DBPROCESS is watching for.                                  | 274         |
| dbresults       | Sets up the results of the next query.   | 275         |
| dbretdata       | Returns a pointer to a return parameter value generated by a stored procedure.                             | 278         |
| dbretlen        | Determines the length of a return parameter value generated by a stored procedure.                         | 282         |
| dbretname       | Determines the name of the stored procedure parameter associated with a particular return parameter value. | 283         |



| <b>Routines</b>  | <b>Description</b>   | <b>Page</b> |
|------------------|--|-------------|
| dbretstatus      | Determines the stored procedure status number returned by the current command or remote procedure call.          | 285         |
| dbrettype        | Determines the datatype of a return parameter value generated by a stored procedure.                             | 287         |
| DBROWS           | Indicates whether the current command actually returned rows.  | 289         |
| DBROWTYPE        | Returns the type of the current row.   | 289         |
| dbrpcinit        | Initializes a remote procedure call.   | 290         |
| dbrpcparam       | Adds a parameter to a remote procedure call.   | 292         |
| dbrpcsend        | Signals the end of a remote procedure call.  | 294         |
| dbrpwclr         | Clears all remote passwords from the LOGINREC structure.   | 295         |
| dbrpwset         | Adds a remote password to the LOGINREC structure.  | 296         |
| dbsafestr        | Doubles the quotes in a character string.  | 297         |
| dbsechandle      | Installs user functions to handle secure logins.   | 299         |
| dbsendpassthru   | Sends a TDS packet to a server.  | 303         |
| dbservcharset    | Gets the name of the server character set.   | 305         |
| dbsetavail       | Marks a DBPROCESS as being available for general use.  | 306         |
| dbsetbusy        | Calls a user-supplied function when DB-Library is reading from the server.                                       | 306         |
| dbsetconnect     | Sets the server connection information.  | 309         |
| dbsetdefcharset  | Sets the default character set for an application.   | 310         |
| dbsetdeflang     | Sets the default language name for an application.   | 311         |
| dbsetidle        | Calls a user-supplied function when DB-Library is finished reading from the server.                              | 312         |
| dbsetifile       | Specifies the name and location of the Sybase interfaces file.   | 313         |
| dbsetinterrupt   | Calls user-supplied functions to handle interrupts while waiting on a read from the server.                      | 314         |
| DBSETLAPP        | Sets the application name in the LOGINREC structure.   | 317         |
| DBSETLCHARSET    | Sets the character set in the LOGINREC structure.  | 318         |
| DBSETLENCRYPT    | Specifies whether or not network password encryption is to be used when logging into Adaptive Server Enterprise. | 319         |
| DBSETLHOST       | Sets the host name in the LOGINREC structure.  | 320         |
| DBSETLMUTUALAUTH | Enables or disables mutual authentication of the connection's security mechanism.                                | 321         |
| DBSETLNATLANG    | Sets the national language name in the LOGINREC structure.   | 322         |

---

| <b>Routines</b>       | <b>Description</b>  | <b>Page</b> |
|-----------------------|---|-------------|
| DBSETLNETWORKAUTH     | Enables or disables network-based authentication.   | 322         |
| dbsetloginfo          | Transfers TDS login information from a DBLOGININFO structure to a LOGINREC structure.                           | 323         |
| dbsetlogintime        | Sets the number of seconds that DB-Library waits for a server response to a request for a DBPROCESS connection. | 325         |
| DBSETLPACKET          | Sets the TDS packet size in an application's LOGINREC structure.  | 326         |
| DBSETLPWD             | Sets the user server password in the LOGINREC structure.  | 327         |
| DBSETLSERVERPRINCIPAL | SSets the server's principal name.  | 328         |
| DBSETLUSER            | Sets the user name in the LOGINREC structure.   | 329         |
| dbsetmaxprocs         | Sets the maximum number of simultaneously open DBPROCESS structures.  | 329         |
| dbsetnull             | Defines substitution values to be used when binding null values.  | 330         |
| dbsetopt              | Sets a server or DB-Library option.   | 332         |
| dbsetrow              | Sets a buffered row to "current."   | 334         |
| dbsettime             | Sets the number of seconds that DB-Library will wait for a server response to a SQL command.                    | 336         |
| dbsetuserdata         | Uses a DBPROCESS structure to save a pointer to user-allocated data.  | 336         |
| dbsetversion          | Specifies a DB-Library version level.   | 339         |
| dbspid                | Gets the server process ID for the specified DBPROCESS.   | 340         |
| dbspr1row             | Places one row of server query results into a buffer.   | 341         |
| dbspr1rowlen          | Determines how large a buffer to allocate to hold the results returned by dbsprhead, dbsprline, and dbspr1row.  | 343         |
| dbsprhead             | Places the server query results header into a buffer.   | 344         |
| dbsprline             | Gets a formatted string that contains underlining for the column names produced by dbsprhead.                   | 346         |
| dbsqlxec              | Sends a command batch to the server.  | 347         |
| dbsqllok              | Waits for results from the server and verify the correctness of the instructions the server is responding to.   | 349         |
| dbsqlsend             | Sends a command batch to the server and do not wait for a response.   | 354         |
| dbstrbuild            | Builds a printable string from text containing placeholders for variables.                                      | 355         |
| dbstrcmp              | Compares two character strings using a specified sort order.  | 358         |
| dbstrcpy              | Copies all or a portion of the command buffer.  | 359         |

| Routines      | Description  | Page |
|---------------|--|------|
| dbstrlen      | Returns the length, in characters, of the command buffer.  | 361  |
| dbstrsort     | Determines which of two character strings should appear first in a sorted list.                        | 362  |
| dbtabbrowse   | Determines whether the specified table is updatable using the DB-Library browse-mode facilities.       | 363  |
| dbtabcount    | Returns the number of tables involved in the current select query.                                     | 364  |
| dbtabname     | Returns the name of a table based on its number.   | 365  |
| dbtabsource   | Returns the name and number of the table from which a particular result column was derived.            | 366  |
| DBTDS         | Determines which version of TDS (the Tabular Data Stream protocol) is being used.                      | 368  |
| dbtextsize    | Returns the number of bytes of text or image data that remain to be read for the current row.          | 368  |
| dbtsnewlen    | Returns the length of the new value of the <i>timestamp</i> column after a browse-mode update.         | 369  |
| dbtsnewval    | Returns the new value of the <i>timestamp</i> column after a browse-mode update.                       | 370  |
| dbtsput       | Puts the new value of the <i>timestamp</i> column into the given table's current row in the DBPROCESS. | 371  |
| dbtxptr       | Returns the value of the text pointer for a column in the current row.                                 | 372  |
| dbtxtimestamp | Returns the value of the text timestamp for a column in the current row.                               | 374  |
| dbtxtsnewval  | Returns the new value of a text timestamp after a call to <code>dbwritetext</code> .                   | 375  |
| dbtxtsput     | Puts the new value of a text timestamp into the specified column of the current row in the DBPROCESS.  | 375  |
| dbuse         | Uses a particular database.  | 376  |
| dbvarylen     | Determines whether the specified regular result column's data can vary in length.                      | 377  |
| dbversion     | Determines which version of DB-Library is in use.  | 378  |
| dbwillconvert | Determines whether a specific datatype conversion is available within DB-Library.                      | 379  |
| dbwritepage   | Writes a page of binary data to the server.  | 381  |
| dbwritetext   | Sends a text or image value to the server.   | 382  |
| dbxlate       | Translates a character string from one character set to another.                                       | 387  |

| Routines | Description  | Page |
|----------|--|------|
| Errors   | The complete collection of DB-Library errors and error severities. | 389  |
| Options  | The complete list of DB-Library options.                           | 407  |
| Types    | Datatypes and symbolic constants for datatypes used by DB-Library. | 412  |

## db12hour

|              |   |
|--------------|---|
| Description  | Determine whether the specified language uses 12-hour or 24-hour time.  |
| Syntax       | <pre>DBBOOL db12hour(dbproc, language)</pre><br><pre>DBPROCESS *dbproc;<br/>char *language;</pre>   |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>language</b><br/>The name of the language of interest.</p>   |
| Return value | “TRUE” if <i>language</i> uses 12-hour time, “FALSE” otherwise.   |
| Usage        | <ul style="list-style-type: none"><li>db12hour returns “TRUE” if <i>language</i> uses 12-hour time, and “FALSE” if it uses 24-hour time.</li><li>If <i>language</i> is NULL, <i>dbproc</i>’s current language is signified. If both <i>language</i> and <i>dbproc</i> are NULL, then DB-Library’s default language (for any future calls to dbopen) is signified.</li><li>db12hour is useful when retrieving and manipulating DBDATETIME values using dbsqlxexec. When converting DBDATETIME values to character strings, dbconvert and dbbind always return the month component of the DBDATETIME value in the local language, but use the U.S. English date and time order (month-day-year, 12-hour time). db12hour’s return value informs the application that some further manipulation is necessary if 24-hour rather than 12-hour time is desired.</li><li>The following code fragment illustrates the use of db12hour:<pre>DBBOOL time_format;</pre></li></ul> |

```
DBCHAR s_date[40];

/*
** Find out whether 12-hour or 24-hour time is
** used.
*/
time_format = db12hour(dbproc, "FRANCAIS");

/* Put a command into a command buffer */
dbcmd(dbproc, "select start_date from info_table");

/* Send the command to the Adaptive Server
Enterprise */
dbsqlxexec(dbproc);

/* Process the command results */
dbresults(dbproc);

/*
** Bind column data (start_date) to the program
** variable (s_date)
*/
dbbind(dbproc, 1, NTBSTRINGBIND, 0, s_date);

while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
    /*
    ** If we want 24-hour time, re-format
    ** s_date accordingly.
    */
    if (time_format == TRUE)
        format_24(s_date);

    printf("Next start date: %s\n", s_date);
}
```

See also `dbdateorder`, `dbdayname`, `dbmonthname`, `dbsetopt`

## dbadata

Description                      Return a pointer to the data for a compute column.

Syntax                            `BYTE *dbadata(dbproc, computeid, colnum)`

```
DBPROCESS *dbproc;  
int computeid;  
int colnum;
```

**Parameters****dbproc**

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

**computeid**

The ID that identifies the particular compute row of interest. A SQL select statement may have multiple compute clauses, each of which returns a separate compute row. The *computeid* corresponding to the first compute clause in a select is 1. The *computeid* is returned by *dbnextrow* or *dbgetrow*.

**colnum**

The number of the column of interest. The first column returned is number 1. Note that the order in which compute columns are returned is determined by the order of the corresponding columns in the select list, not by the order in which the compute columns were originally specified. For example, in the following query the result of “sum(price)” is referenced by giving *colnum* a value of 1, not 2:

```
select price, advance from titles  
compute sum(advance), sum(price)
```

The relative order of compute columns in the select list, rather than their absolute position, determines the value of *colnum*. For instance, given the following variation of the previous select:

```
select title_id, price, advance from titles  
compute sum(advance), sum(price)
```

the *colnum* for “sum(price)” still has a value of 1 and not 2, because the “title\_id” column in the select list is not a compute column and therefore is ignored when determining the compute column’s number.

**Return value**

A BYTE pointer to the data for a particular column in a particular compute. Be sure to cast this pointer into the proper type. A BYTE pointer to NULL is returned if there is no such column or compute or if the data has a null value.

DB-Library allocates and frees the data space that the BYTE pointer points to. Do not overwrite this space.

**Usage**

- After each call to *dbnextrow*, you can use this routine to return a pointer to the data for a particular column in a compute row. The data is not null-terminated. You can use *dbadlen* to get the length of the data.

- When a column of integer data is summed or averaged, the server always returns a 4-byte integer, regardless of the size of the column. Therefore, be sure that the variable that is to contain the result from such a compute is declared as DBINT.
- Here is a short program fragment which illustrates the use of dbadata:

```

DBPROCESS      *dbproc;
int             rowinfo;
DBINT           sum;

/*
** First, put the commands into the command
** buffer
*/
dbcmd(dbproc, "select db_name(dbid), dbid, size
              from sysusages");
dbcmd(dbproc, " order by dbid");
dbcmd(dbproc, " compute sum(size) by dbid");

/*
** Send the commands to Adaptive Server Enterprise
and start
** execution
*/
dbsqlxexec(dbproc);

/* Process the command */
dbresults(dbproc);

/* Examine the results of the compute clause */
while((rowinfo = dbnextrow(dbproc)) !=
      NO_MORE_ROWS)
{
    if (rowinfo == REG_ROW)
        printf("regular row returned.\n");
    else
    {
        /*
        ** This row is the result of a compute
        ** clause, and "rowinfo" is the computeid
        ** of this compute clause.
        */

        sum = *(DBINT *) (dbadata(dbproc, rowinfo,
                                  1));
        printf("sum = %ld\n", sum);
    }
}

```

```
    }  
}
```

- The function dbaltbind automatically binds compute data to your program variables. It does a copy of the data, but is often easier to use than dbadata. Furthermore, it includes a convenient type conversion capability. By means of this capability, the application can, among other things, easily add a null terminator to a result string or convert money and datetime data to printable strings.

See also

dbadlen, dbaltbind, dbaltlen, dbalttype, dbgetrow, dbnextrow, dbnumalts

## dbadlen

Description

Return the actual length of the data for a compute column.

Syntax

DBINT dbadlen(dbproc, computeid, column)

```
DBPROCESS  *dbproc;  
int         computeid;  
int         column;
```

Parameters

dbproc

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

computeid

The ID that identifies the particular compute row of interest. A SQL select statement may have multiple compute clauses, each of which returns a separate compute row. The *computeid* corresponding to the first compute clause in a select is 1. The *computeid* is returned by dbnextrow or dbgetrow.

column

The number of the column of interest. The first column is number 1.

Return value

The length, in bytes, of the data for a particular compute column. If there is no such column or compute clause, dbadlen returns -1. If the data has a null value, dbadlen returns 0.

Usage

- This routine returns the actual length of the data for a particular compute column.



- Use the `dbattlen` routine to determine the maximum possible length for the data. Use `dbadata` to get a pointer to the data.
- Here is a program fragment that illustrates the use of `dbadlen`:

```

DBPROCESS      *dbproc;
char            biggest_name[MAXNAME+1];
int             namelen;
int             rowinfo;

/* put the command into the command buffer */
dbcmd(dbproc, "select name from sysobjects");
dbcmd(dbproc, " order by name");
dbcmd(dbproc, " compute max(name)");

/*
** Send the command to Adaptive Server Enterprise
and start
** execution.
*/
dbsqlxexec(dbproc);

/* process the command */
dbresults(dbproc);

/* examine each row returned by the command */
while ((rowinfo = dbnextrow(dbproc)) !=
       NO_MORE_ROWS)
{
    if (rowinfo == REG_ROW)
        printf("regular row returned.\n");
    else
    {
        /*
        ** This row is the result of a compute
        ** clause, and "rowinfo" is the computeid
        ** of this compute clause.
        */
        namelen = dbadlen(dbproc, rowinfo, 1);
        strncpy(biggest_name,
            (char *)dbadata(dbproc, rowinfo, 1),
            namelen);

        /*
        ** Data pointed to by dbadata() is not
        ** null-terminated.
        */
    }
}

```

```
        biggest_name[namelen] = '\0';

        printf("biggest name = %s\n",
            biggest_name);
    }
}
```

See also

dbadata, dbaltlen, dbalttype, dbgetrow, dbnextrow, dbnumalts

## dbaltbind

Description

Bind a compute column to a program variable.

Syntax

```
RETCODE dbaltbind(dbproc, computeid, column, vartype,
    varlen, varaddr)
```

```
DBPROCESS    *dbproc;
int           computeid;
int           column;
int           vartype;
DBINT        varlen;
BYTE         * varaddr;
```

Parameters

**dbproc**

A pointer to the DBPROCESS structure that provides the connection for a particular front end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

**computeid**

The ID that identifies the particular compute row of interest. A select statement may have multiple compute clauses, each of which returns a separate compute row. The *computeid* corresponding to the first compute clause in a select is 1.

**column**

The column number of the row data that is to be copied to a program variable. The first column is column number 1. Note that the order in which compute columns are returned is determined by the order of the corresponding columns in the select list, not by the order in which the compute columns were originally specified. For example, in the following query the result of “sum(price)” is referenced by giving *column* a value of 1, not 2:

```
select price, advance from titles
```

```
compute sum(advance), sum(price)
```

The relative order of compute columns in the select list, rather than their absolute position, determines the value of *column*. For instance, given the following variation of the earlier select:

```
select title_id, price, advance from titles
compute sum(advance), sum(price)
```

the *column* for “sum(price)” still has a value of 1 and not 2, because the “title\_id” column in the select list is not a compute column and therefore is ignored when determining the compute column’s number.

#### *vartype*

This describes the datatype of the binding. It must correspond to the datatype of the program variable that will receive the copy of the data from the DBPROCESS. The table below shows the correspondence between *vartype* values and program variable types.

dbaltbind supports a wide range of type conversions, so the *vartype* can be different from the type returned by the SQL query. For instance, a SYBMONEY result may be bound to a DBFLT8 program variable through FLT8BIND, and the appropriate data conversion will happen automatically. For a list of the data conversions provided by DB-Library, see the reference page for dbwillconvert.

---

**Note** dbaltbind does not offer explicit precision and scale support for numeric and decimal datatypes. When handling numeric or decimal data, dbaltbind uses a default precision and scale of 18 and 0, respectively, unless the bind is to a numeric or decimal column, in which case dbaltbind uses the precision and scale of the source data. Use dbaltbind\_ps to explicitly specify precision and scale values—calling dbaltbind is equivalent to calling dbaltbind\_ps with a NULL *typeinfo* value.

---

For a list of the type definitions used by DB-Library, see Types on page 412.

Table 2-1 lists the legal *vartype* values recognized by dbaltbind, along with the server and program variable types that each one refers to:

**Table 2-1: Bind types (dbaltbind)**

| <b>Vartype</b>    | <b>Program variable type</b> | <b>Server datatype</b> |
|-------------------|------------------------------|------------------------|
| CHARBIND          | DBCHAR                       | SYBCHAR                |
| STRINGBIND        | DBCHAR                       | SYBCHAR                |
| NTBSTRINGBIND     | DBCHAR                       | SYBCHAR                |
| VARYCHARBIND      | DBVARYCHAR                   | SYBCHAR                |
| BINARYBIND        | DBBINARY                     | SYBBINARY              |
| VARYBINBIND       | DBVARYBIN                    | SYBBINARY              |
| TINYBIND          | DBTINYINT                    | SYBINT1                |
| SMALLBIND         | DBSMALLINT                   | SYBINT2                |
| INTBIND           | DBINT                        | SYBINT4                |
| FLT8BIND          | DBFLT8                       | SYBFLT8                |
| REALBIND          | DBREAL                       | SYBREAL                |
| NUMERICBIND       | DBNUMERIC                    | SYBNUMERIC             |
| DECIMALBIND       | DBDECIMAL                    | SYBDECIMAL             |
| BITBIND           | DBBIT                        | SYBBIT                 |
| DATETIMEBIND      | DBDATETIME                   | SYBDATETIME            |
| SMALLDATETIMEBIND | DBDATETIME4                  | SYBDATETIME4           |
| MONEYBIND         | DBMONEY                      | SYBMONEY               |
| SMALLMONEYBIND    | DBMONEY4                     | SYBMONEY4              |
| BOUNDARYBIND      | DBCHAR                       | SYBBOUNDARY            |
| SENSITIVITYBIND   | DBCHAR                       | SYBSENSITIVITY         |

---

**Warning!** It is an error to use any of the following values for *vartype* if the library version has not been set (with `dbsetversion`) to `DBVERSION_100` or higher: `BOUNDARYBIND`, `DECIMALBIND`, `NUMERICBIND`, or `SENSITIVITYBIND`.

---

Since `SYBTEXT` and `SYBIMAGE` data are never returned through a compute row, those datatypes are not listed above.

Note that the server type in the table above is listed merely for your information. The *vartype* you specify does not necessarily have to correspond to a particular server type, because, as mentioned earlier, `dbaltbind` will convert server data into the specified *vartype*.

The available representations for character data are shown below. They differ according to whether the data is blank-padded or null-terminated:

| Vartype         | Program type | Padding | Terminator |
|-----------------|--------------|---------|------------|
| CHARBIND        | DBCHAR       | blanks  | none       |
| STRINGBIND      | DBCHAR       | blanks  | \0         |
| NTBSTRINGBIND   | DBCHAR       | none    | \0         |
| VARYCHARBIND    | DBVARYCHAR   | none    | none       |
| BOUNDARYBIND    | DBCHAR       | none    | \0         |
| SENSITIVITYBIND | DBCHAR       | none    | \0         |

Note that the “\0” in the table above is the null terminator character.

If overflow occurs when converting integer or float data to a character binding type, the first character of the resulting value will contain an asterisk (“\*”) to indicate the error.

Binary data may be stored in two different ways:

| Vartype     | Program Type | Padding |
|-------------|--------------|---------|
| BINARYBIND  | DBBINARY     | nulls   |
| VARYBINBIND | DBVARBINARY  | none    |

When a column of integer data is summed or averaged, the server always returns a 4-byte integer, regardless of the size of the column. Therefore, be sure that the variable which is to contain the result from such a compute is declared as DBINT and that the *vartype* of the binding is INTBIND.

#### varlen

The length of the program variable in bytes.

For *vartype* values that represent fixed-length types, such as MONEYBIND or FLT8BIND, this length is ignored.

For character and binary types, *varlen* must describe the total length of the available destination buffer space, including any space that may be required for special terminating bytes, such as a null terminator. If *varlen* is 0, the total number of bytes available will be copied into the program variable. (For char and binary server data, the total number of bytes available is equal to the defined length of the database column, including any blank padding. For varchar and varbinary data, the total number of bytes available is equal to the actual data contained in the column.) Therefore, if you are sure that your program variable is large enough to handle the results, you can just set *varlen* to 0.

#### varaddr

The address of the program variable to which the data will be copied.

## Return value

SUCCEED or FAIL.

dbaltbind returns FAIL if the column number is not valid, if the data conversion specified by *vartype* is not legal, or if *varaddr* is NULL.

## Usage

- This routine directs DB-Library to copy compute column data returned by the server into a program variable. (A compute column results from the compute clause of a Transact-SQL select statement.) When each new row containing compute data is read using *dbnextrow* or *dbgetrow*, the data from the designated *column* in that compute row is copied into the program variable with the address *varaddr*. There must be a separate *dbaltbind* call for each compute column that is to be copied. It is not necessary to bind every compute column to a program variable.
- The server can return two types of rows: regular rows containing data from columns designated by a select statement's select list, and compute rows resulting from the compute clause. *dbaltbind* binds data from compute rows. Use *dbbind* for binding data from regular rows.
- You must make the calls to *dbaltbind* after a call to *dbresults* and before the first call to *dbnextrow*.
- The typical sequence of calls is:

```
DBCHAR    name[20];
DBINT     namecount;
```

```
/* read the query into the command buffer */
dbcmd(dbproc, "select name from emp compute
count(name)");
```

```
/* send the query to Adaptive Server Enterprise */
dbsqlxec(dbproc);
```

```
/* get ready to process the query results */
dbresults(dbproc);
```

```
/* bind the regular row data (name) */
dbbind(dbproc, 1, STRINGBIND, (DBINT) 0, name);
```

```
/* bind the compute column data (count of name) */
dbaltbind(dbproc, 1, 1, INTBIND, (DBINT) 0,
(BYTE *) &namecount);
```

```
/* now process each row */
while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
```

```
    C-code to print or process row data
```

```
}
```

- dbaltbind incurs a little overhead because it causes the data to be copied into a program variable. To avoid this copying, you can use the dbadata routine to directly access the returned data.
- You can only bind a result column to a single program variable. If you bind a result column to multiple variables, only the last binding takes effect.
- The server can return null column values, and DB-Library provides the following aids for handling null values:
  - A pre-defined set of default values, one for each datatype, that DB-Library automatically substitutes when a bound column contains a null value. The dbsetnull function allows you to explicitly set your own null substitution values. See the reference page for the dbsetnull function for a list of the default substitution values.
  - The ability to bind an indicator variable to a column with dbnullbind (or dbanullbind for compute rows). As rows are fetched, the value of the indicator variable will be set to indicate whether or not the column value was null. See the reference page for the dbnullbind function for indicator values and meanings.

See also

dbadata, dbaltbind\_ps, dbanullbind, dbbind, dbbind\_ps, dbconvert, dbconvert\_ps, dbnullbind, dbsetnull, dbsetversion, dbwillconvert, Types on page 412

## dbaltbind\_ps

|             |  |
|-------------|--|
| Description | Bind a compute column to a program variable, with precision and scale support for numeric and decimal datatypes.   |
| Syntax      | <pre>RETCODE dbaltbind_ps(dbproc, computeid, column,                     vartype, varlen, varaddr,                     typeinfo)</pre><br><pre>DBPROCESS  *dbproc; int         computeid; int         column; int         vartype; DBINT      varlen; BYTE       *varaddr; DBTYPEINFO *typeinfo;</pre> |

## Parameters

## dbproc

A pointer to the DBPROCESS structure that provides the connection for a particular front end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

## computeid

The ID that identifies the particular compute row of interest. A select statement may have multiple compute clauses, each of which returns a separate compute row. The *computeid* corresponding to the first compute clause in a select is 1.

## column

The column number of the row data that is to be copied to a program variable. The first column is column number 1. Note that the order in which compute columns are returned is determined by the order of the corresponding columns in the select list, not by the order in which the compute columns were originally specified. For example, in the following query the result of “sum(price)” is referenced by giving *column* a value of 1, not 2:

```
select price, advance from titles
compute sum(advance), sum(price)
```

The relative order of compute columns in the select list, rather than their absolute position, determines the value of *column*. For instance, given the following variation of the earlier select:

```
select title_id, price, advance from titles
compute sum(advance), sum(price)
```

the *column* for “sum(price)” still has a value of 1 and not 2, because the “title\_id” column in the select list is not a compute column and therefore is ignored when determining the compute column’s number.



**vartype**

This describes the datatype of the binding. It must correspond to the datatype of the program variable that will receive the copy of the data from the DBPROCESS. The table below shows the correspondence between *vartype* values and program variable types.

dbaltbind\_ps supports a wide range of type conversions, so the *vartype* can be different from the type returned by the SQL query. For instance, a SYBMONEY result may be bound to a DBFLT8 program variable through FLT8BIND, and the appropriate data conversion will happen automatically. For a list of the data conversions provided by DB-Library, see the reference page for dbwillconvert.

---

**Note** dbaltbind\_ps's parameters are identical to dbaltbind's, except that dbaltbind\_ps has the additional parameter *typeinfo*, which contains information about precision and scale for DBNUMERIC or DBDECIMAL variables.

---

For a list of the type definitions used by DB-Library, see Types on page 412.

Table 2-2 lists the legal *vartype* values recognized by dbaltbind\_ps, along with the server and program variable types that each one refers to:

**Table 2-2: Bind types (dbaltbind\_ps)**

| <b>Vartype</b>    | <b>Program variable type</b> | <b>Server datatype</b> |
|-------------------|------------------------------|------------------------|
| CHARBIND          | DBCHAR                       | SYBCHAR                |
| STRINGBIND        | DBCHAR                       | SYBCHAR                |
| NTBSTRINGBIND     | DBCHAR                       | SYBCHAR                |
| VARYCHARBIND      | DBVARYCHAR                   | SYBCHAR                |
| BINARYBIND        | DBBINARY                     | SYBBINARY              |
| VARYBINBIND       | DBVARYBIN                    | SYBBINARY              |
| TINYBIND          | DBTINYINT                    | SYBINT1                |
| SMALLBIND         | DBSMALLINT                   | SYBINT2                |
| INTBIND           | DBINT                        | SYBINT4                |
| FLT8BIND          | DBFLT8                       | SYBFLT8                |
| REALBIND          | DBREAL                       | SYBREAL                |
| NUMERICBIND       | DBNUMERIC                    | SYBNUMERIC             |
| DECIMALBIND       | DBDECIMAL                    | SYBDECIMAL             |
| BITBIND           | DBBIT                        | SYBBIT                 |
| DATETIMEBIND      | DBDATETIME                   | SYBDATETIME            |
| SMALLDATETIMEBIND | DBDATETIME4                  | SYBDATETIME4           |
| MONEYBIND         | DBMONEY                      | SYBMONEY               |
| SMALLMONEYBIND    | DBMONEY4                     | SYBMONEY4              |
| BOUNDARYBIND      | DBCHAR                       | SYBBOUNDARY            |
| SENSITIVITYBIND   | DBCHAR                       | SYBSENSITIVITY         |

---

**Warning!** It is an error to use any of the following values for *vartype* if the library version has not been set (with `dbsetversion`) to `DBVERSION_100` or higher: `BOUNDARYBIND`, `DECIMALBIND`, `NUMERICBIND`, or `SENSITIVITYBIND`.

---

Since `SYBTEXT` and `SYBIMAGE` data are never returned through a `compute row`, those datatypes are not listed above.

Note that the server type in the table above is listed merely for your information. The *vartype* you specify does not necessarily have to correspond to a particular server type, because, as mentioned earlier, `dbaltbind_ps` will convert server data into the specified *vartype*.

The available representations for character data are shown below. They differ according to whether the data is blank-padded or null-terminated:

| Vartype         | Program type | Padding | Terminator |
|-----------------|--------------|---------|------------|
| CHARBIND        | DBCHAR       | blanks  | none       |
| STRINGBIND      | DBCHAR       | blanks  | \0         |
| NTBSTRINGBIND   | DBCHAR       | none    | \0         |
| VARYCHARBIND    | DBVARYCHAR   | none    | none       |
| BOUNDARYBIND    | DBCHAR       | none    | \0         |
| SENSITIVITYBIND | DBCHAR       | none    | \0         |

Note that the “\0” in the table above is the null terminator character.

If overflow occurs when converting integer or float data to a character binding type, the first character of the resulting value will contain an asterisk (“\*”) to indicate the error.

Binary data may be stored in two different ways:

| Vartype     | Program type | Padding |
|-------------|--------------|---------|
| BINARYBIND  | DBBINARY     | nulls   |
| VARYBINBIND | DBVARBINARY  | none    |

When a column of integer data is summed or averaged, the server always returns a 4-byte integer, regardless of the size of the column. Therefore, be sure that the variable which is to contain the result from such a compute is declared as DBINT and that the *vartype* of the binding is INTBIND.

#### varlen

The length of the program variable in bytes.

For values of *vartype* that represent a fixed-length type, such as MONEYBIND or FLT8BIND, this length is ignored.

For character and binary types, *varlen* must describe the total length of the available destination buffer space, including any space that may be required for special terminating bytes, such as a null terminator. If *varlen* is 0, the total number of bytes available will be copied into the program variable. (For char and binary server data, the total number of bytes available is equal to the defined length of the database column, including any blank padding. For varchar and varbinary data, the total number of bytes available is equal to the actual data contained in the column.) Therefore, if you are sure that your program variable is large enough to handle the results, you can just set *varlen* to 0.

#### varaddr

The address of the program variable to which the data will be copied.

**typeinfo**

A pointer to a DBTYPEINFO structure containing information about the precision and scale of decimal or numeric data. An application sets a DBTYPEINFO structure with values for precision and scale before calling dbaltbind\_ps to bind columns to DBDECIMAL or DBNUMERIC variables.

If *typeinfo* is NULL:

- If the result column is of type numeric or decimal, dbaltbind\_ps picks up precision and scale values from the result column.
- If the result column is not numeric or decimal, dbaltbind\_ps uses a default precision of 18 and a default scale of 0.

If *vartype* is not DECIMALBIND or NUMERICBIND, *typeinfo* is ignored.

A DBTYPEINFO structure is defined as follows:

```
typedef struct typeinfo {  
    DBINT    precision;  
    DBINT    scale;  
} DBTYPEINFO;
```

Legal values for *precision* are from 1 to 77. Legal values for *scale* are from 0 to 77. *scale* must be less than or equal to *precision*.

**Return value**

SUCCEED or FAIL.

dbaltbind\_ps returns FAIL if the column number is not valid, if the data conversion specified by *vartype* is not legal, or if *varaddr* is NULL.

**Usage**

- dbaltbind\_ps is the equivalent of dbaltbind, except that dbaltbind\_ps provides precision and scale support for numeric and decimal datatypes, which dbaltbind does not. Calling dbaltbind is equivalent to calling dbaltbind\_ps with *typeinfo* as NULL.
- dbaltbind\_ps directs DB-Library to copy compute column data returned by the server into a program variable. (A compute column results from the compute clause of a Transact-SQL select statement.) When each new row containing compute data is read using dbnextrow or dbgetrow, the data from the designated *column* in that compute row is copied into the program variable with the address *varaddr*. There must be a separate dbaltbind\_ps call for each compute column that is to be copied. It is not necessary to bind every compute column to a program variable.
- The server can return two types of rows: regular rows containing data from columns designated by a select statement's select list, and compute rows resulting from the compute clause. dbaltbind\_ps binds data from compute rows. Use dbbind\_ps for binding data from regular rows.

- You must make the calls to `dbaltbind_ps` after a call to `dbresults` and before the first call to `dbnextrow`.
- `dbaltbind_ps` incurs some overhead because it causes the data to be copied into a program variable. To avoid this copying, you can use the `dbadata` routine to directly access the returned data.
- You can only bind a result column to a single program variable. If you bind a result column to multiple variables, only the last binding takes effect.
- Since the server can return null values, DB-Library provides a set of default values, one for each datatype, that it will automatically substitute when binding null values. The `dbsetnull` function allows you to explicitly set your own null substitution values. (See the reference page for the `dbsetnull` function for a list of the default substitution values.)

See also

`dbaltbind`, `dbanullbind`, `dbbind`, `dbbind_ps`, `dbconvert`, `dbconvert_ps`, `dbdata`, `dbnullbind`, `dbsetnull`, `dbsetversion`, `dbwillconvert`, Types on page 412

## dbaltcolid

Description

Return the column ID for a compute column.

Syntax

```
int dbaltcolid(dbproc, computeid, column)
```

```
DBPROCESS  *dbproc;
int         computeid;
int         column;
```

Parameters

`dbproc`

A pointer to the `DBPROCESS` structure that provides the connection for a particular front end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

`computeid`

The ID that identifies the particular compute row of interest. A SQL select statement may have multiple compute clauses, each of which returns a separate compute row. The *computeid* corresponding to the first compute clause in a select is 1. The *computeid* is returned by `dbnextrow` or `dbgetrow`.

`column`

The number of the compute column of interest. The first column in a select list is 1.

|              |  |
|--------------|--|
| Return value | The select list ID for the compute column. The first column in a select list is 1. If either the <i>computeid</i> or the <i>column</i> value is invalid, dbaltcolid returns -1.  |
| Usage        | <ul style="list-style-type: none"><li>This routine returns the select list ID for a compute column. For example, given the SQL statement:<pre>select dept, name from employee order by dept, name compute count(name) by dept</pre>the call dbaltcolid(<i>dbproc</i>, 1, 1) will return 2, since “name” is the second column in the select list.</li></ul> |
| See also     | dbadata, dbadlen, dbaltlen, dbgetrow, dbnextrow, dbnumalts, dbrtype  |

## dbaltlen

|              |   |
|--------------|---|
| Description  | Return the maximum length of the data for a particular compute column.  |
| Syntax       | <pre>DBINT dbaltlen(dbproc, computeid, column)</pre><br><pre>DBPROCESS  *dbproc; int         computeid; int         column;</pre>   |
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>computeid</b></p> <p>The ID that identifies the particular compute row of interest. A SQL select statement may have multiple compute clauses, each of which returns a separate compute row. The <i>computeid</i> corresponding to the first compute clause in a select is 1. The <i>computeid</i> is returned by dbnextrow or dbgetrow.</p> <p><b>column</b></p> <p>The number of the column of interest. The first column is number 1.</p> |
| Return value | The maximum length, in bytes, possible for the data in a particular compute column. dbaltlen returns -1 if there is no such column or compute clause.   |
| Usage        | This routine returns the maximum length for a column in a compute row. In the case of variable length data, this is not necessarily the actual length of the data, but rather the maximum length. For the actual data length, use dbadlen.  |

For example, given the SQL statement:

```
select dept, name from employee
order by dept, name
compute count(name) by dept
```

the call `dbaltlen(dbproc, 1, 1)` returns 4 because counts are of SYBINT4 type, which is 4 bytes long.

See also

`dbadata`, `dbadlen`, `dbalttype`, `dbgetrow`, `dbnextrow`, `dbnumalts`

## dbaltop

|              |  |
|--------------|--|
| Description  | Return the type of aggregate operator for a particular compute column.   |
| Syntax       | <code>int dbaltop(dbproc, computeid, column)</code>  |
|              | <pre>DBPROCESS  *dbproc; int         computeid; int         column;</pre>  |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>computeid</b><br/>The ID that identifies the particular compute row of interest. A SQL select statement may have multiple compute clauses, each of which returns a separate compute row. The <i>computeid</i> corresponding to the first compute clause in a select is 1. The <i>computeid</i> is returned by <code>dbnextrow</code> or <code>dbgetrow</code>.</p> <p><b>column</b><br/>The number of the column of interest. The first column is number 1.</p> |
| Return value | A token value for the type of the compute column's aggregate operator. In case of error, <code>dbaltop</code> returns -1.  |
| Usage        | <ul style="list-style-type: none"> <li>This routine returns the type of aggregate operator for a particular column in a compute row. For example, given the SQL statement: <pre>select dept, name from employee order by dept, name compute count(name) by dept</pre> </li> </ul>  |

the call `dbaltop(dbproc, 1, 1)` will return the token value for count since the first aggregate operator in the first compute clause is count.

- You can convert the token value to a readable token string with `dbprtype`. See the `dbprtype` reference page for a list of all token values and their equivalent token strings.

See also

`dbadata`, `dbadlen`, `dbaltlen`, `dbnextrow`, `dbnumalts`, `dbprtype`

## dbaltype

Description

Return the datatype for a compute column.

Syntax

```
int dbaltype(dbproc, computeid, column)
```

```
DBPROCESS    *dbproc;  
int           computeid;  
int           column;
```

Parameters

`dbproc`

A pointer to the `DBPROCESS` structure that provides the connection for a particular front end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

`computeid`

The ID that identifies the particular compute row of interest. A SQL select statement may have multiple compute clauses, each of which returns a separate compute row. The *computeid* corresponding to the first compute clause in a select is 1. The *computeid* is returned by `dbnextrow` or `dbgetrow`.

`column`

The number of the column of interest. The first column is number 1.

Return value

A token value for the datatype for a particular compute column.

In a few cases, the token value returned by this routine may not correspond exactly with the column's server datatype:

- `SYBVARCHAR` is returned as `SYBCHAR`.
- `SYBVARBINARY` is returned as `SYBBINARY`.
- `SYBDATETIME` is returned as `SYBDATETIME`.
- `SYBMONEYN` is returned as `SYBMONEY`.



- SYBFLTN is returned as SYBFLT8.
- SYBINTN is returned as SYBINT1, SYBINT2, or SYBINT4, depending on the actual type of the SYBINTN.

dbaltype returns -1 if either the *computeid* or the *column* value is invalid.

#### Usage

- This routine returns the datatype for a compute column. For a list of server datatypes, see Types on page 412.
- dbaltype actually returns an integer token value for the datatype (SYBCHAR, SYBFLT8, and so on). To convert the token value into a readable token string, use dbprtype. See the dbprtype reference page for a list of all token values and their equivalent token strings.
- For example, given the SQL statement:

```
select dept, name from employee
order by dept, name
compute count(name) by dept
```

the call `dbaltype(dbproc, 1, 1)` returns the token value SYBINT4, because counts are of SYBINT4 type. `dbprtype` will convert SYBINT4 into the readable token string “int”.

#### See also

dbadata, dbadlen, dbaltlen, dbnextrow, dbnumalts, dbprtype, Types on page 412

## dbaltutype

#### Description

Return the user-defined datatype for a compute column.

#### Syntax

DBINT dbaltutype(dbproc, computeid, column)

```
DBPROCESS  *dbproc;
int         computeid;
int         column;
```

#### Parameters

dbproc

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

|              |  |
|--------------|--|
|              | <b>computeid</b><br>The ID that identifies the particular compute row of interest. A SQL select statement may have multiple compute clauses, each of which returns a separate compute row. The <i>computeid</i> corresponding to the first compute clause in a select is 1. The <i>computeid</i> is returned by dbnextrow or dbgetrow.   |
|              | <b>column</b><br>The number of the column of interest. The first column is number 1.   |
| Return value | The user-defined datatype of the specified compute column on success; a negative integer on error.   |
| Usage        | <ul style="list-style-type: none"><li>• dbaltutype returns the user-defined datatype for a compute column.</li><li>• For a description of how to add user-defined datatypes to the server databases or Server-Library programs, see the <i>Adaptive Server Enterprise Reference Manual</i> or the <i>Open Server Server-Library/C Reference Manual</i>.</li><li>• dbaltutype is defined as type DBINT, since both the DB-Library datatype DBINT and user-defined datatypes are 32 bits long.</li></ul> |
| See also     | dbalttype, dbcolutype  |

## dbanullbind

|             |  |
|-------------|--|
| Description | Associate an indicator variable with a compute-row column.   |
| Syntax      | RETCODE dbanullbind(dbproc, computeid, column,<br>indicator)<br><br>DBPROCESS *dbproc;<br>int computeid;<br>int column;<br>DBINT *indicator;   |
| Parameters  | <b>dbproc</b><br>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server. |

**computeid**

The compute row of interest. A select statement may have multiple compute clauses, each of which returns a separate compute row. The *computeid* corresponding to the first compute clause in a select is 1.

**column**

The number of the column that is to be associated with the indicator variable.

**indicator**

A pointer to the indicator variable.

---

**Note** *indicator* is just the pointer to the indicator variable. It is the variable itself that is set.

---

**Return value**

SUCCEED or FAIL.

`dbnullbind` returns FAIL if either *computeid* or *column* is invalid.

**Usage**

- `dbnullbind` associates a compute-row column with an indicator variable. The indicator variable indicates whether a particular compute-row column has been converted and copied to a program variable successfully or unsuccessfully, or whether it is null.
- The indicator variable is set when compute rows are processed using `dbnextrow`. The possible values are:
  - *-1* if the column is NULL.
  - *The full length of the column's data, in bytes* if the column was bound to a program variable using `dbaltbind`, the binding did not specify any data conversions, and the bound data was truncated because the program variable was too small to hold the column's data.
  - *0* if the column was bound and copied to a program variable successfully.

---

**Note** Detection of character string truncation is implemented only for CHARBIND and VARYCHARBIND.

---

**See also**

`dbadata`, `dbadlen`, `dbaltbind`, `dbnextrow`, `dbnullbind`

## dbbind

|             |  |
|-------------|--|
| Description | Bind a regular result column to a program variable.  |
| Syntax      | <pre>RETCODE ddbbind(dbproc, column, vartype, varlen,<br/>                varaddr)</pre><br><pre>DBPROCESS    *dbproc;<br/>int           column;<br/>int           vartype;<br/>DBINT        varlen;<br/>BYTE         *varaddr;</pre>  |
| Parameters  | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>column</b><br/>The column number of the row data that is to be copied to a program variable. The first column is column number 1.</p> |

**vartype**

This describes the datatype of the binding. It must correspond to the datatype of the program variable that will receive the copy of the data from the DBPROCESS. The following table shows the correspondence between *vartype* values and program variable types.

dbbind supports a wide range of type conversions, so the *vartype* can be different from the type returned by the SQL query. For example, a SYBMONEY result may be bound to a DBFLT8 program variable through FLT8BIND, and the appropriate data conversion will happen automatically. For a list of the data conversions provided by DB-Library, see the reference page for dbwillconvert.

---

**Note** The ddbind routine does not offer explicit precision and scale support for numeric and decimal datatypes. When handling numeric or decimal data, ddbind uses a default precision and scale of 18 and 0, respectively, unless the bind is to a numeric or decimal column, in which case ddbind uses the precision and scale of the source data. Use ddbind\_ps to explicitly specify precision and scale values—calling ddbind is equivalent to calling ddbind\_ps with a NULL *typeinfo* value.

---

For a list of the type definitions used by DB-Library, see Types on page 412.

Table 2-3 lists the legal *vartype* values recognized by ddbind, along with the server and program variable types that each one refers to:

**Table 2-3: Bind types (dbbind)**

| Vartype           | Program variable type | Server datatype       |
|-------------------|-----------------------|-----------------------|
| CHARBIND          | DBCHAR                | SYBCHAR or SYBTEXT    |
| STRINGBIND        | DBCHAR                | SYBCHAR or SYBTEXT    |
| NTBSTRINGBIND     | DBCHAR                | SYBCHAR or SYBTEXT    |
| VARYCHARBIND      | DBVARYCHAR            | SYBCHAR or SYBTEXT    |
| BINARYBIND        | DBBINARY              | SYBBINARY or SYBIMAGE |
| VARYBINBIND       | DBVARYBIN             | SYBBINARY or SYBIMAGE |
| TINYBIND          | DBTINYINT             | SYBINT1               |
| SMALLBIND         | DBSMALLINT            | SYBINT2               |
| INTBIND           | DBINT                 | SYBINT4               |
| FLT8BIND          | DBFLT8                | SYBFLT8               |
| REALBIND          | DBREAL                | SYBREAL               |
| NUMERICBIND       | DBNUMERIC             | SYBNUMERIC            |
| DECIMALBIND       | DBDECIMAL             | SYBDECIMAL            |
| BITBIND           | DBBIT                 | SYBBIT                |
| DATETIMEBIND      | DBDATETIME            | SYBDATETIME           |
| SMALLDATETIMEBIND | DBDATETIME4           | SYBDATETIME4          |
| MONEYBIND         | DBMONEY               | SYBMONEY              |
| SMALLMONEYBIND    | DBMONEY4              | SYBMONEY4             |
| BOUNDARYBIND      | DBCHAR                | SYBBOUNDARY           |
| SENSITIVITYBIND   | DBCHAR                | SYBSENSITIVITY        |

---

**Warning!** An error occurs when you use any of the following values for *vartype* if the library version has not been set (with `dbsetversion`) to `DBVERSION_100` or higher: `BOUNDARYBIND`, `DECIMALBIND`, `NUMERICBIND`, or `SENSITIVITYBIND`.

---

The server type in the table above is listed merely for your information. The *vartype* you specify does not necessarily have to correspond to a particular server type, because, as mentioned earlier, `dbbind` will convert server data into the specified *vartype*.

---

**Note** The server types `nchar` and `nvarchar` are converted internally to `char` and `varchar` types, which correspond to the DB-Library type constant `SYBCHAR`.

---

The available representations for character and text data are shown below.

They differ according to whether the data is blank-padded or null-terminated. Note that if *varlen* is 0, no padding takes place and that the “\0” is the null terminator character:

| Vartype         | Program type | Padding | Terminator |
|-----------------|--------------|---------|------------|
| CHARBIND        | DBCHAR       | blanks  | none       |
| STRINGBIND      | DBCHAR       | blanks  | \0         |
| NTBSTRINGBIND   | DBCHAR       | none    | \0         |
| VARYCHARBIND    | DBVARYCHAR   | none    | none       |
| BOUNDARYBIND    | DBCHAR       | none    | \0         |
| SENSITIVITYBIND | DBCHAR       | none    | \0         |

If overflow occurs when converting integer or float data to a character/text binding type, the first character of the resulting value will contain an asterisk (“\*”) to indicate the error.

Binary and image data can be stored in two different ways:

| Vartype     | Program type | Padding |
|-------------|--------------|---------|
| BINARYBIND  | DBBINARY     | nulls   |
| VARYBINBIND | DBVARBINARY  | none    |

*varlen*

The length of the program variable in bytes.

For values of *vartype* that represent a fixed-length type, such as *MONEYBIND* or *FLT8BIND*, this length is ignored.

For char, text, binary, and image types, *varlen* must describe the total length of the available destination buffer space, including any space that may be required for special terminating bytes, such as a null terminator. If *varlen* is 0, the total number of bytes available will be copied into the program variable. (For char and binary server data, the total number of bytes available is equal to the defined length of the database column, including any blank padding. For varchar, varbinary, text, and image data, the total number of bytes available is equal to the actual data contained in the column.) Therefore, if you are sure that your program variable is large enough to handle the results, you can just set *varlen* to 0.

Note that if *varlen* is 0, no padding takes place.

In some cases, DB-Library issues a message indicating that data conversion resulted in an overflow. This can be caused by a *varlen* specification that is too small for the server data.

*varaddr*

The address of the program variable to which the data will be copied.

Return value

SUCCEED or FAIL.

*dbbind* returns FAIL if the column number is not valid, if the data conversion specified by *vartype* is not legal, or if *varaddr* is NULL.

Usage

- Data comes back from the server one row at a time. This routine directs DB-Library to copy the data for a regular column (designated in a select statement's select list) into a program variable. When each new row containing regular (*not* compute) data is read using *dbnextrow* or *dbgetrow*, the data from the designated *column* in that row is copied into the program variable with the address *varaddr*. There must be a separate *dbbind* call for each regular column that is to be copied. It is not necessary to bind every column to a program variable.
- The server can return two types of rows: regular rows and compute rows resulting from the compute clause of a select statement. *dbbind* binds data from regular rows. Use *dbaltbind* for binding data from compute rows.
- You must make the calls to *dbbind* after a call to *dbresults* and before the first call to *dbnextrow*.
- The typical sequence of calls is:

```
DBINT      xvariable;
DBCHAR     yvariable[10];

/* read the query into the command buffer */
dbcmd(dbproc, "select x = 100, y = 'hello'");

/* send the query to Adaptive Server Enterprise */
dbsqlxexec(dbproc);

/* get ready to process the query results */
dbresults(dbproc);

/* bind column data to program variables */
dbbind(dbproc, 1, INTBIND, (DBINT) 0,
      (BYTE *) &xvariable);
dbbind(dbproc, 2, STRINGBIND, (DBINT) 0,
      yvariable);

/* now process each row */
while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
    C-code to print or process row data
}
```



```
}

```

- `dbbind` incurs a little overhead, because it causes the data to be copied into a program variable. To avoid this copying, you can use the `dbdata` routine to directly access the returned data.
- You can only bind a result column to a single program variable. If you bind a result column to multiple variables, only the last binding takes effect.
- Since the server can return null values, DB-Library provides a set of default values, one for each datatype, that it will automatically substitute when binding null values. The `dbsetnull` function allows you to explicitly set your own null substitution values. (See the reference page for the `dbsetnull` function for a list of the default substitution values.)

See also

`dbaltbind`, `dbaltbind_ps`, `dbnullbind`, `dbbind_ps`, `dbconvert`, `dbconvert_ps`, `dbdata`, `dbnullbind`, `dbsetnull`, `dbsetversion`, `dbwillconvert`, Types on page 412

## dbbind\_ps

Description

Bind a regular result column to a program variable, with precision and scale support for numeric and decimal datatypes.

Syntax

```
RETCODE ddbbind_ps(dbproc, column, vartype, varlen,
                   varaddr, typeinfo)
```

```
DBPROCESS    *dbproc;
int           column;
int           vartype;
DBINT         varlen;
BYTE          *varaddr;
DBTYPEINFO    *typeinfo;
```

Parameters

`dbproc`

A pointer to the `DBPROCESS` structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

`column`

The column number of the row data that is to be copied to a program variable. The first column is column number 1.

#### vartype

This describes the datatype of the binding. It must correspond to the datatype of the program variable that will receive the copy of the data from the DBPROCESS. The table below shows the correspondence between *vartype* values and program variable types.

dbbind\_ps supports a wide range of type conversions, so the *vartype* can be different from the type returned by the SQL query. For instance, a SYBMONEY result may be bound to a DBFLT8 program variable through FLT8BIND, and the appropriate data conversion will happen automatically. For a list of the data conversions provided by DB-Library, see the reference page for dbwillconvert.

For a list of the type definitions used by DB-Library, see Types on page 412.

Table 2-4 lists the legal *vartype* values recognized by ddbind\_ps, along with the server and program variable types that each one refers to:

**Table 2-4: Bind types (*dbbind\_ps*)**

| <b>Vartype</b>    | <b>Program variable type</b> | <b>Server type</b>    |
|-------------------|------------------------------|-----------------------|
| CHARBIND          | DBCHAR                       | SYBCHAR or SYBTEXT    |
| STRINGBIND        | DBCHAR                       | SYBCHAR or SYBTEXT    |
| NTBSTRINGBIND     | DBCHAR                       | SYBCHAR or SYBTEXT    |
| VARYCHARBIND      | DBVARYCHAR                   | SYBCHAR or SYBTEXT    |
| BINARYBIND        | DBBINARY                     | SYBBINARY or SYBIMAGE |
| VARYBINBIND       | DBVARYBIN                    | SYBBINARY or SYBIMAGE |
| TINYBIND          | DBTINYINT                    | SYBINT1               |
| SMALLBIND         | DBSMALLINT                   | SYBINT2               |
| INTBIND           | DBINT                        | SYBINT4               |
| FLT8BIND          | DBFLT8                       | SYBFLT8               |
| REALBIND          | DBREAL                       | SYBREAL               |
| NUMERICBIND       | DBNUMERIC                    | SYBNUMERIC            |
| DECIMALBIND       | DBDECIMAL                    | SYBDECIMAL            |
| BITBIND           | DBBIT                        | SYBBIT                |
| DATETIMEBIND      | DBDATETIME                   | SYBDATETIME           |
| SMALLDATETIMEBIND | DBDATETIME4                  | SYBDATETIME4          |
| MONEYBIND         | DBMONEY                      | SYBMONEY              |
| SMALLMONEYBIND    | DBMONEY4                     | SYBMONEY4             |
| BOUNDARYBIND      | DBCHAR                       | SYBBOUNDARY           |
| SENSITIVITYBIND   | DBCHAR                       | SYBSENSITIVITY        |

**Warning!** It is an error to use any of the following values for *vartype* if the library version has not been set (with *dbsetversion*) to *DBVERSION\_100* or higher: *BOUNDARYBIND*, *DECIMALBIND*, *NUMERICBIND*, or *SENSITIVITYBIND*.\*

The server type in the table above is listed merely for your information. The *vartype* you specify does not necessarily have to correspond to a particular server type, because, as mentioned earlier, *dbbind\_ps* will convert server data into the specified *vartype*.

**Note** The server types *nchar* and *nvarchar* are converted internally to *char* and *varchar* types, which correspond to the DB-Library type constant *SYBCHAR*.

The available representations for character and text data are shown below.

They differ according to whether the data is blank-padded or null-terminated. Note that if *varlen* is 0, no padding takes place and that the “\0” is the null terminator character:

| Vartype         | Program type | Padding | Terminator |
|-----------------|--------------|---------|------------|
| CHARBIND        | DBCHAR       | blanks  | none       |
| STRINGBIND      | DBCHAR       | blanks  | \0         |
| NTBSTRINGBIND   | DBCHAR       | none    | \0         |
| VARYCHARBIND    | DBVARYCHAR   | none    | none       |
| BOUNDARYBIND    | DBCHAR       | none    | \0         |
| SENSITIVITYBIND | DBCHAR       | none    | \0         |

If overflow occurs when converting integer or float data to a character/text binding type, the first character of the resulting value will contain an asterisk (“\*”) to indicate the error.

binary and image data may be stored in two different ways:

| Vartype     | Program variable type | Padding |
|-------------|-----------------------|---------|
| BINARYBIND  | DBBINARY              | nulls   |
| VARYBINBIND | DBVARBINARY           | none    |

*varlen*

The length of the program variable in bytes.

For values of *vartype* that represent a fixed-length type, such as *MONEYBIND* or *FLT8BIND*, this length is ignored.

For char, text, binary, and image types, *varlen* must describe the total length of the available destination buffer space, including any space that may be required for special terminating bytes, such as a null terminator. If *varlen* is 0, the total number of bytes available will be copied into the program variable. (For char and binary server data, the total number of bytes available is equal to the defined length of the database column, including any blank padding. For varchar, varbinary, text, and image data, the total number of bytes available is equal to the actual data contained in the column.)

Therefore, if you are sure that your program variable is large enough to handle the results, you can just set *varlen* to 0.

---

**Note** If *varlen* is 0, no padding takes place.

---

*varaddr*

The address of the program variable to which the data will be copied.

**typeinfo**

A pointer to a DBTYPEINFO structure containing information about the precision and scale of decimal or numeric data. An application sets a DBTYPEINFO structure with values for precision and scale before calling dbbind\_ps to bind columns to DBDECIMAL or DBNUMERIC variables.

If *typeinfo* is NULL:

- If the result column is of type numeric or decimal, dbbind\_ps picks up precision and scale values from the result column.
- If the result column is not numeric or decimal, dbbind\_ps uses a default precision of 18 and a default scale of 0.

If *vartype* is not DECIMALBIND or NUMERICBIND, *typeinfo* is ignored.

A DBTYPEINFO structure is defined as follows:

```
typedef struct typeinfo {
    DBINTprecision;
    DBINTscale;
} DBTYPEINFO;
```

Legal values for *precision* are from 1 to 77. Legal values for *scale* are from 0 to 77. *scale* must be less than or equal to *precision*.

**Return value**

SUCCEED or FAIL.

dbbind\_ps returns FAIL if the column number is not valid, if the data conversion specified by *vartype* is not legal, or if *varaddr* is NULL.

**Usage**

- dbbind\_ps parameters are identical to dbbind's, except that dbbind\_ps has the additional parameter *typeinfo*, which contains information about precision and scale for DBNUMERIC or DBDECIMAL variables.
- dbbind\_ps is the equivalent of dbbind, except that dbbind\_ps provides scale and precision support for numeric and decimal datatypes, which dbbind does not. Calling dbbind is equivalent to calling dbbind\_ps with *typeinfo* as NULL.
- Data comes back from the server one row at a time. This routine directs DB-Library to copy the data for a regular column (designated in a select statement's select list) into a program variable. When each new row containing regular (*not* compute) data is read using dbnextrow or dbgetrow, the data from the designated *column* in that row is copied into the program variable with the address *varaddr*. There must be a separate dbbind or dbbind\_ps call for each regular column that is to be copied. It is not necessary to bind every column to a program variable.

- The server can return two types of rows: regular rows and compute rows resulting from the compute clause of a select statement. Use `dbbind_ps` to bind data from regular rows, and `dbaltbind_ps` to bind data from compute rows.
- You must make the calls to `dbbind_ps` after a call to `dbresults` and before the first call to `dbnextrow`.
- `dbbind_ps` incurs some overhead, because it causes the data to be copied into a program variable. To avoid this copying, you can use the `dbdata` routine to directly access the returned data.
- You can bind a result column only to a single program variable. If you bind a result column to multiple variables, only the last binding takes effect.
- Since the server can return null values, DB-Library provides a set of default values, one for each datatype, that it will automatically substitute when binding null values. The `dbsetnull` function allows you to explicitly set your own null substitution values. See the reference page for the `dbsetnull` function for a list of the default substitution values.

See also

`dbaltbind`, `dbaltbind_ps`, `dbanullbind`, `dbbind`, `dbconvert`, `dbconvert_ps`, `dbdata`, `dbnullbind`, `dbsetnull`, `dbsetversion`, `dbwillconvert`, [Types on page 412](#)

## dbbufsize

Description

Return the size of a DBPROCESS row buffer.

Syntax

```
int dbbufsize(dbproc)
```

DBPROCESS      \*dbproc;

Parameters

`dbproc`

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.

Return value

An integer representing the size, in rows, of the DBPROCESS row buffer.

If *dbproc* is NULL or if row buffering is not allowed, `dbbufsize` returns 0.

Usage

- `dbbufsize` returns the size of a DBPROCESS row buffer.

- Row buffering provides a way for an application to keep a specified number of server result rows in program memory. To allow row buffering, call `dbsetopt(dbproc, DBBUFFER, n)`, where *n* is the number of rows to buffer. An application that is buffering result rows can access rows non-sequentially, using `dbgetrow`. See the `dbgetrow` reference page for a discussion of the benefits and penalties of row buffering.

See also `dbclrbuf`, `dbgetrow`, `dbsetopt`, Options on page 407

## dbbylist

Description Return the bylist for a compute row.

Syntax `BYTE *dbbylist(dbproc, computeid, size)`

```
DBPROCESS  *dbproc;
int         computeid;
int         *size;
```

Parameters `dbproc`

A pointer to the `DBPROCESS` structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

`computeid`

The ID that identifies the particular compute row of interest. A SQL select statement may have multiple compute clauses, each of which returns a separate compute row. The *computeid* corresponding to the first compute clause in a select is 1. The *computeid* is returned by `dbnextrow` or `dbgetrow`.

`size`

A pointer to an integer, which `dbbylist` sets to the number of elements in the bylist.

Return value

A pointer to an array of bytes containing the numbers of the columns that compose the bylist for the specified compute. The array of BYTES is part of the `DBPROCESS`, so you must not free it. If the *computeid* is out of range, NULL is returned.

Call `dbcolname` to derive the name of a column from its number.

The size of the array is returned in the *size* parameter. A *size* of 0 indicates that either there is no bylist for this particular compute or the *computeid* is out of range.

|          |   |
|----------|---|
| Usage    | <ul style="list-style-type: none"><li>• dbbylist returns the bylist for a compute row. (A select statement's compute clause may contain the keyword by, followed by a list of columns. This list, known as the "bylist," divides the results into subgroups, based on changing values in the specified columns. The compute clause's row aggregate is applied to each subgroup, generating a compute row for each subgroup.)</li><li>• dbresults must return SUCCEED before the application calls this routine.</li><li>• Assume the following command has been executed:<br/><pre>select dept, name, year, sales from employee order by dept, name, year compute count(name) by dept,name</pre><p>The call dbbylist(<i>dbproc</i>, 1, &amp;<i>size</i>) sets <i>size</i> to 2, because there are two items in the bylist. It returns a pointer to an array of two BYTEs, which contain the values 1 and 2, indicating that the bylist is composed of columns 1 and 2 from the select list.</p></li></ul> |
| See also | dbadata, dbadlen, dbaltlen, dbalttype, dbcolname, dbgetrow, dbnextrow   |

## dbcancel

|              |   |
|--------------|---|
| Description  | Cancel the current command batch.   |
| Syntax       | RETCODE dbcancel( <i>dbproc</i> )   |
| Parameters   | <p>DBPROCESS    *<i>dbproc</i>;</p> <p><i>dbproc</i></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> |
| Return value | <p>SUCCEED or FAIL.</p> <p>The most common reasons for failure are a dead DBPROCESS or a network error. dbcancel will also return FAIL if the server is dead.</p>   |



|          |   |
|----------|---|
| Usage    | <ul style="list-style-type: none"><li>• This routine cancels execution of the current command batch on the server and flushes any pending results. The application can call it after calling <code>dbsqlxexec</code>, <code>dbsqlsend</code>, <code>dbsqllok</code>, <code>dbresults</code>, or <code>dbnextrow</code>. The <code>dbcancel</code> routine sends an attention packet to the server which causes the server to cease execution of the command batch. Any pending results are read and discarded.</li><li>• <code>dbcancel</code> cancels <i>all</i> the commands in the current command batch. To cancel only the results from the current command, call <code>dbcancelquery</code> instead.</li><li>• Some applications may need the ability to cancel a long-running query while DB-Library is reading from the network. In this case, the application should use one of these methods:<ul style="list-style-type: none"><li>• Set a time limit for server reads with <code>dbsettime</code>, and add a special case to your error handler function to respond to SYBETIME errors. See the reference pages for <code>dberrhandle</code> and <code>dbsettime</code> for details.</li><li>• Use <code>dbsetinterrupt</code> to install custom interrupt handling. See the reference page for <code>dbsetinterrupt</code> for details.</li></ul></li><li>• If you have set your own interrupt handler using <code>dbsetinterrupt</code>, you cannot call <code>dbcancel</code> in your interrupt handler. This would cause output from the server to DB-Library to become out of sync. See the reference page for <code>dbsetinterrupt</code> for an explanation of how to cancel from an interrupt handler.</li></ul> |
| See also | <code>dbcancelquery</code> , <code>dbnextrow</code> , <code>dbresults</code> , <code>dbsetinterrupt</code> , <code>dbsqlxexec</code> , <code>dbsqllok</code> , <code>dbsqlsend</code>   |

## dbcancelquery

|              |  |
|--------------|--|
| Description  | Cancel any rows pending from the most recently executed query.   |
| Syntax       | <pre>RETCODE dbcancelquery(dbproc)</pre><br><pre>DBPROCESS *dbproc;</pre>  |
| Parameters   | <p><code>dbproc</code></p> <p>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> |
| Return value | SUCCEED or FAIL.   |

The most common reasons for failure are a dead DBPROCESS or a network error.

|          |  |
|----------|--|
| Usage    | <ul style="list-style-type: none"><li>• This routine is an efficient way to throw away any unread rows that result from the most recently executed SQL query. Calling <code>dbcanquery</code> is equivalent to calling <code>dbnextrow</code> until it returns <code>NO_MORE_ROWS</code>, but <code>dbcanquery</code> is faster because it allocates no memory and executes no bindings to user data.</li><li>• If you have set your own interrupt handler using <code>dbsetinterrupt</code>, you cannot call <code>dbcanquery</code> in your interrupt handler. This would cause output from the server to DB-Library to become out of sync. If you want to ignore any unread rows from the current query, the interrupt handler should set a flag that you can check before the next call to <code>dbnextrow</code>.</li><li>• <code>dbresults</code> must return <code>SUCCEED</code> before an application can call <code>dbcanquery</code>.</li><li>• If you want to ignore all of the results from all of the commands in the current command batch, call <code>dbcancel</code> instead.</li></ul> |
| See also | <code>dbcancel</code> , <code>dbnextrow</code> , <code>dbresults</code> , <code>dbsetinterrupt</code> , <code>dbsqlxexec</code>  |

## dbchange

|              |   |
|--------------|---|
| Description  | Determine whether a command batch has changed the current database.   |
| Syntax       | <pre>char *dbchange(dbproc)</pre><br><pre>DBPROCESS  *dbproc;</pre>   |
| Parameters   | <p><code>dbproc</code></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> |
| Return value | A pointer to the null-terminated name of the new database, if any. If the database has not changed, NULL will be returned.  |
| Usage        | <ul style="list-style-type: none"><li>• <code>dbchange</code> informs the program of a change in the current database. It does so by catching any instance of the Transact-SQL <code>use</code> command.</li></ul>  |

- Although a use command can appear anywhere in a command batch, the database change does not actually take effect until the end of the batch. `dbchange` is therefore useful only in determining whether the current command batch has changed the database for subsequent command batches.
- The internal `DBPROCESS` flag that `dbchange` monitors to determine whether the database has changed is cleared when the program executes a new command batch by calling either `dbsqlxexec` or `dbsqlsend`. Therefore, the simplest way to keep track of database changes is to call `dbchange` when `dbresults` returns `NO_MORE_RESULTS` at the end of each command batch.
- Alternatively, you can always get the name of the current database by calling `dbname`.

See also `dbname`, `dbresults`, `dbsqlxexec`, `dbsqlsend`, `dbuse`

## dbcharsetconv

|              |  |
|--------------|--|
| Description  | Indicate whether the server is performing character set translation.   |
| Syntax       | <code>DBBOOL dbcharsetconv(dbproc)</code><br><br><code>DBPROCESS *dbproc;</code>   |
| Parameters   | <code>dbproc</code><br>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and the server.  |
| Return value | “TRUE” if the server is performing character set translations; “FALSE” if it is not.   |
| Usage        | <ul style="list-style-type: none"><li>• If a client and a server are using the same character set, the server is not performing translation. In this case, <code>dbcharsetconv</code> returns “FALSE”.</li><li>• To get the name of its own character set, a client can call <code>dbgetcharset</code>.</li><li>• To get the name of the server’s character set, a client can call <code>dbservcharset</code>.</li></ul> |
| See also     | <code>dbservcharset</code> , <code>dbgetcharset</code> , <code>DBSETLCHARSET</code>  |

## **dbclose**

|              |  |
|--------------|--|
| Description  | Close and deallocate a single DBPROCESS structure.   |
| Syntax       | <pre>void dbclose(dbproc)</pre><br><pre>DBPROCESS  *dbproc;</pre>  |
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>  |
| Return value | None.  |
| Usage        | <ul style="list-style-type: none"><li>• <code>dbclose</code> is the inverse of <code>dbopen</code>. It cleans up any activity associated with one DBPROCESS structure and deallocates the space. It also closes the corresponding network connection.</li><li>• To close every open DBPROCESS structure, use <code>dbexit</code> instead.</li><li>• <code>dbclose</code> does not deallocate space associated with a LOGINREC. To deallocate a LOGINREC, an application can call <code>dbloginfree</code>.</li><li>• Calling <code>dbclose</code> with an argument not returned by <code>dbopen</code> is sure to cause trouble.</li></ul> |
| See also     | <code>dbexit</code> , <code>dbopen</code>  |

## **dbclrbuf**

|             |   |
|-------------|---|
| Description | Drop rows from the row buffer.  |
| Syntax      | <pre>void dbclrbuf(dbproc, n)</pre><br><pre>DBPROCESS* dbproc;<br/>DBINT      n;</pre>  |
| Parameters  | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> |

*n*

The number of rows you want cleared from the row buffer. If you make *n* equal to or greater than the number of rows in the buffer, all but the newest row will be removed. If *n* is less than 1, the function call is ignored.

Return value

None.

Usage

- DB-Library provides a row-buffering service to application programs. You can turn row buffering on by calling `dbsetopt(dbproc, DBBUFFER, n)` where *n* is the number of rows you would like DB-Library to buffer. If buffering is on, you can then randomly refer to rows that have been read from the server, using `dbgetrow`. See the `dbgetrow` reference page for a discussion of the benefits and penalties of row buffering.
- The row buffer can become full for two reasons. Either the server has returned more than the *n* rows you said you wanted buffered, or sufficient space could not be allocated to save the row you wanted. When the row buffer is full, `dbnextrow` returns `BUF_FULL` and refuses to read in the next row from the server. Once the row buffer is full, subsequent calls to `dbnextrow` will continue to return `BUF_FULL` until at least one row is freed by calling `dbclrbuf`. `dbclrbuf` always frees the oldest rows in the buffer first.
- Once a result row has been cleared from the buffer, it is no longer available to the program.
- For an example of row buffering, see the sample program *example4.c*.

See also

`dbgetrow`, `dbnextrow`, `dbsetopt`, Options on page 407

## dbclropt

Description

Clear an option set by `dbsetopt`.

Syntax

`RETCODE dbclropt(dbproc, option, param)`

```
DBPROCESS  *dbproc;
int         option;
char*       param;
```

## Parameters

**dbproc**

A pointer to the DBPROCESS structure that provides the connection for a particular front end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server. If *dbproc* is NULL, the option will be cleared for all active DBPROCESS structures.

**option**

The option that is to be turned off. See Options on page 407 for a list of options.

**param**

Certain options take parameters. The DBOFFSET option, for example, takes as a parameter the SQL construct for which offsets are to be returned. Options on page 407 lists those options that take parameters. If an option does not take a parameter, *param* must be NULL.

If the option you are clearing takes a parameter, but there can be only one instance of the option, dbclropt ignores the *param* argument. For example, dbclropt ignores the value of *param* when clearing the DBBUFFER option, because row buffering can have only one setting at a time. On the other hand, the DBOFFSET option can have several settings, each with a different parameter. It may have been set twice—to look for offsets to select statements and offsets to order by clauses. In that case, dbclropt needs the *param* argument to determine whether to clear the select offset or the order by offset.

If an invalid parameter is specified for one of the server options, this will be discovered the next time a command buffer is sent to the server. The dbsqlexec or dbsqlsend call fails, and DB-Library will invoke the user-installed message handler. If an invalid parameter is specified for one of the DB-Library options (DBBUFFER or DBTEXTLIMIT), the dbclropt call itself fails.

## Return value

SUCCEED or FAIL.

## Usage

- This routine clears the server and DB-Library options that have been set with dbsetopt. Although server options may be set and cleared directly through SQL, the application should instead use dbsetopt and dbclropt to set and clear options. This provides a uniform interface for setting both server and DB-Library options. It also allows the application to use the dbisopt function to check the status of an option.
- dbclropt does not immediately clear the option. The option is cleared the next time a command buffer is sent to the server (by invoking dbsqlexec or dbsqlsend).

- For a complete list of options, see Options on page 407.

See also

dbisopt, dbsetopt, Options on page 407

## dbcmd

Description

Add text to the DBPROCESS command buffer.

Syntax

RETCODE dbcmd(dbproc, cmdstring)

```
DBPROCESS *dbproc;
char      *cmdstring;
```

Parameters

dbproc

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

cmdstring

A null-terminated character string that dbcmd copies into the command buffer.

Return value

SUCCEED or FAIL.

Usage

- This routine adds text to the Transact-SQL command buffer in the DBPROCESS structure. It adds to the existing command buffer—it does not delete or overwrite the current contents except after the buffer has been sent to the server (see “Clearing the command buffer” on page 92). A single command buffer may contain multiple commands; in fact, this represents an efficient use of the command buffer.
- dbfcmd is a related function. dbfcmd interprets the *cmdstring* as a format string that is passed to `sprintf` along with any additional arguments. The application can intermingle calls to dbcmd and dbfcmd.

Consecutive calls to dbcmd

- The application may call dbcmd repeatedly. The command strings in sequential calls are just concatenated together. It is the application’s responsibility to ensure that any necessary blanks appear between the end of one string and the beginning of the next.

- Here is a small example of using `dbcmd` to build up a multiline SQL command:

```
DBPROCESS      *dbproc;

    dbcmd(dbproc, "select name from sysobjects");
    dbcmd(dbproc, " where id < 5");
    dbcmd(dbproc, " and type='S'");
```

Note the required spaces at the start of the second and third command strings.

- At any time, the application can access the contents of the command buffer through calls to `dbgetchar`, `dbstrlen`, and `dbstrcpy`.
- Available memory is the only constraint on the size of the `DBPROCESS` command buffer created by calls to `dbcmd` and `dbfcmd`.

Clearing the command buffer

- After a call to `dbsqlxexec` or `dbsqlsend`, the first call to either `dbcmd` or `dbfcmd` automatically clears the command buffer before the new text is entered. If this situation is undesirable, set the `DBNOAUTOFREE` option. When `DBNOAUTOFREE` is set, the command buffer is cleared only by an explicit call to `dbfreebuf`.

See also

`dbfcmd`, `dbfreebuf`, `dbgetchar`, `dbstrcpy`, `dbstrlen`, Options on page 407

## DBCMDROW

|              |  |
|--------------|--|
| Description  | Determine whether the current command can return rows.   |
| Syntax       | RETCODE DBCMDROW(dbproc)   |
| Parameters   | <p>DBPROCESS    *dbproc;</p> <p>dbproc</p> <p>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> |
| Return value | SUCCEED or FAIL, to indicate whether the command can return rows.  |



|          |  |
|----------|--|
| Usage    | <ul style="list-style-type: none"> <li>• DBCMDROW determines whether the command currently being processed by dbresults is one that can return rows—that is, a Transact-SQL select statement or an execute on a stored procedure containing a select. The application can call it after dbresults returns SUCCEED.</li> <li>• Even if DBCMDROW macro returns SUCCEED, the command does not return any rows if none have qualified. To determine whether any rows are actually being returned, use DBROWS.</li> </ul> |
| See also | dbnextrow, dbresults, DBROWS, DBROWTYPE  |

## dbcolbrowse

|              |  |
|--------------|--|
| Description  | Determine whether the source of a regular result column is updatable through the DB-Library browse-mode facilities.  |
| Syntax       | DBBOOL dbcolbrowse(dbproc, colnum)   |
| Parameters   | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p>colnum</p> <p>The number of the result column of interest. Column numbers start at 1.</p>  |
| Return value | “TRUE” or “FALSE.”   |
| Usage        | <ul style="list-style-type: none"> <li>• dbcolbrowse is one of the DB-Library browse mode routines. See Chapter 1, “Introducing DB-Library” for a detailed discussion of browse mode.</li> <li>• dbcolbrowse provides a way to determine whether the database column that is the source of a regular (that is, non-compute) result column in a select list is updatable using the DB-Library browse-mode facilities. This routine is useful in examining ad hoc queries. If the query has been hard-coded into the program, dbcolbrowse obviously is unnecessary.</li> <li>• To be updatable, a column must be derived from a browsable table (that is, the table must have a unique index and a timestamp column) and cannot be the result of a SQL expression. For example, in the following select list:</li> </ul> |

```
select title, category=type,
```

wholesale=(price \* 0.6) ... for browse

result columns 1 and 2 (“title” and “category”) are updatable, but column 3 (“wholesale”) is not, because it is the result of an expression.

- The application can call dbcolbrowse anytime after dbresults.
- To determine the name of the source column given the name of the result column, use dbcolsource.
- The sample program *example7.c* contains a call to dbcolbrowse.

See also

dbcolsource, dbqual, dbtabbrowse, dbtabcount, dbtabname, dbtabsource, dbtsnewlen, dbtsnewval, dbtspu

## dbcollen

Description

Return the maximum length of the data in a regular result column.

Syntax

DBINT dbcollen(dbproc, column)

DBPROCESS \*dbproc;  
int column;

Parameters

dbproc

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

column

The number of the column of interest. The first column is number 1.

Return value

The maximum length, in bytes, of the data for the particular column. If the column number is not in range, dbcollen returns -1.

Usage

- This routine returns the maximum length of the data in a regular (that is, non-compute) result column. In the case of variable length data, this is not necessarily the actual length of the data, but rather the maximum length that the data can be. For the actual data length, use dbdatlen.

- The value that `dbcollen` returns is not affected by Transact-SQL string functions such as `rtrim` and `ltrim`. For example, if the column `au_lname` has a maximum length of 20 characters, and the first row instance of `au_lname` is “Goodman ” (a value padded with 13 spaces), `dbcollen` returns 20 as the length of `au_lname`, even though the Transact-SQL command `select rtrim(au_lname) from authors` returns a string that is 5 characters long.
- Here is a small program fragment that uses `dbcollen`:

```
DBPROCESS      *dbproc;
int             colnum;
DBINT          column_length;

/* Put the command into the command buffer */
dbcmd(dbproc, "select name, id, type from
              sysobjects");

/*
** Send the command to Adaptive Server Enterprise
and begin
** execution
*/
dbsqlxexec(dbproc);

/* process the command results */
dbresults(dbproc);

/* examine the column lengths */
for (colnum = 1; colnum < 4; colnum++)
{
    column_length = dbcollen(dbproc, colnum);
    printf("column %d, length is %ld.\n", colnum,
          column_length);
}
```

See also `dbcolname`, `dbcoltype`, `dbdata`, `dbdatlen`, `dbnumcols`

## dbcolname

Description                      Return the name of a regular result column.

Syntax                            `char *dbcolname(dbproc, column)`

|              |   |
|--------------|---|
|              | <pre>DBPROCESS      *dbproc; int             column;</pre>  |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>column</b><br/>The number of the column of interest. The first column is number 1.</p>   |
| Return value | A CHAR pointer to the null-terminated name of the particular column. If the column number is not in range, dbcolname returns NULL.  |
| Usage        | <ul style="list-style-type: none"><li>• This routine returns a pointer to the null-terminated name of a regular (that is, non-compute) result column.</li><li>• Here is a small program fragment that uses dbcolname:<pre>DBPROCESS      *dbproc;  /* Put the command into the command buffer */ dbcmd(dbproc, "select name, id, type from                sysobjects");  /* ** Send the command to Adaptive Server Enterprise and begin ** execution */ dbsqlxexec(dbproc);  /* Process the command results */ dbresults(dbproc);  /* Examine the column names */ printf("first column name is %s\n",         dbcolname(dbproc, 1)); printf("second column name is %s\n",         dbcolname(dbproc, 2)); printf("third column name is %s\n",         dbcolname(dbproc, 3));</pre></li></ul> |
| See also     | dbcollen, dbcoltype, dbdata, dbdatlen, dbnumcols  |

## dbcolsource

|              |  |
|--------------|--|
| Description  | Return a pointer to the name of the database column from which the specified regular result column was derived.  |
| Syntax       | <pre>char *dbcolsource(dbproc, colnum)</pre> <pre>DBPROCESS *dbproc; int colnum;</pre>   |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>colnum</b><br/>The number of the result column of interest. Column numbers start at 1.</p>  |
| Return value | A pointer to a null-terminated column name. This pointer will be NULL if the column number is out of range or if the column is the result of a SQL expression, such as max(colname).   |
| Usage        | <ul style="list-style-type: none"> <li>dbcolsource is one of the DB-Library browse mode routines. It is usable only with results from a browse-mode select (that is, a select containing the key words for browse). See Chapter 1, “Introducing DB-Library” for a detailed discussion of browse mode.</li> <li>dbcolsource provides an application with information it needs to update a database column, based on an ad hoc query. select statements may optionally specify header names for regular (that is, non-compute) result columns: <pre>select author = au_lname from authors for browse</pre> <p>When updating a table, you must use the database column name, not the header name (in this example, “au_lname”, not “author”). You can use the dbcolsource routine to get the underlying database column name:</p> <pre>dbcolsource(dbproc, 1)</pre> <p>This call returns a pointer to the string “au_lname”.</p> </li> </ul> <ul style="list-style-type: none"> <li>dbcolsource is useful for ad hoc queries. If the query has been hard-coded into the program, this routine obviously is unnecessary.</li> <li>The application can call dbcolsource anytime after dbresults.</li> <li>The sample program <i>example7.c</i> contains a call to dbcolsource.</li> </ul> |

See also `dbcolbrowse`, `dbqual`, `dbtabbrowse`, `dbtabcount`, `dbtabname`, `dbtabsource`, `dbtsnewlen`, `dbtsnewval`, `dbtsput`

## dbcoltype

**Description** Return the datatype for a regular result column.

**Syntax** `int dbcoltype(dbproc, column)`

```
DBPROCESS  *dbproc;
int         column;
```

**Parameters** `dbproc`  
A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

`column`  
The number of the column of interest. The first column is number 1.

**Return value** A token value for the datatype for a particular column.

In a few cases, the token value returned by this routine may not correspond exactly with the column's server datatype:

- SYBVARCHAR is returned as SYBCHAR.
- SYBVARBINARY is returned as SYBBINARY.
- SYBDATETIMN is returned as SYBDATETIME.
- SYBMONEYN is returned as SYBMONEY.
- SYBFLT8N is returned as SYBFLT8.
- SYBINTN is returned as SYBINT1, SYBINT2, or SYBINT4, depending on the actual type of the SYBINTN.

If the column number is not in range, `dbcoltype` returns -1.

**Usage**

- This routine returns the datatype for a regular (that is, non-compute) result column. For a list of server datatypes, see Types on page 412.

- `dbcoltype` actually returns an integer token value for the datatype (SYBCHAR, SYBFLT8, and so on). To convert the token value into a readable token string, use `dbprtype`. See the `dbprtype` reference page for a list of all token values and their equivalent token strings.
- You can use `dbvarylen` to determine whether a column's datatype is variable length.
- Here is a program fragment that uses `dbcoltype`:

```
DBPROCESS      *dbproc;
int            colnum;
int            coltype;

/* Put the command into the command buffer */
dbcmd(dbproc, "select name, id, type from
             sysobjects");

/* Send the command to Adaptive Server Enterprise
and begin
** execution.
*/
dbsqlxexec(dbproc);

/* Process the command results */
dbresults(dbproc);

/* Examine the column types */
for (colnum = 1; colnum < 4; colnum++)
{
    coltype = dbcoltype(dbproc, colnum);
    printf("column %d, type is %s.\n", colnum,
          dbprtype(coltype));
}
```

See also `dbcollen`, `dbcolname`, `dbdata`, `dbdatlen`, `dbnumcols`, `dbprtype`, `dbvarylen`,  
Types on page 412

## dbcoltypeinfo

**Description** Return precision and scale information for a regular result column of type numeric or decimal.

|              |  |
|--------------|--|
| Syntax       | <code>DBTYPEINFO * dbcoltypeinfo(dbproc, column)</code>  |
|              | <code>DBPROCESS *dbproc;</code><br><code>int column;</code>  |
| Parameters   | <p><code>dbproc</code><br/>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><code>column</code><br/>The number of the column of interest. The first column is number 1.</p>   |
| Return value | <p>A pointer to a <code>DBTYPEINFO</code> structure that contains precision and scale values for a particular numeric or decimal column, or <code>NULL</code> if the specified column number is not in the result set.</p> <p>A <code>DBTYPEINFO</code> structure is defined as follows:</p> <pre>typedef struct typeinfo {<br/>    DBINT    precision;<br/>    DBINT    scale;<br/>} DBTYPEINFO;</pre> <p>If the datatype of the column is not numeric or decimal, the returned structure will contain meaningless values. Check that <code>dbcoltype</code> returns <code>SYBNUMERIC</code> or <code>SYBDECIMAL</code> before calling this function.</p> |
| Usage        | <ul style="list-style-type: none"><li>• This routine returns a pointer to a <code>DBTYPEINFO</code> structure that provides precision and scale information for a regular (that is, non-compute) result column of datatype numeric or decimal.</li><li>• The precision and scale values returned for columns with other datatypes will be meaningless. Check that <code>dbcoltype</code> returns <code>SYBNUMERIC</code> or <code>SYBDECIMAL</code> before calling <code>dbcoltypeinfo</code>.</li></ul>   |
| See also     | <code>dbcollen</code> , <code>dbcolname</code> , <code>dbcoltype</code> , <code>dbdata</code> , <code>dbdatlen</code> , <code>dbnumcols</code> , <code>dbprtype</code> , <code>dbvarylen</code> , Types on page 412  |

## dbcolutype

|             |   |
|-------------|---|
| Description | Return the user-defined datatype for a regular result column. |
| Syntax      | <code>DBINT dbcolutype(dbproc, column)</code>                 |



|              |   |
|--------------|---|
|              | <pre>DBPROCESS *dbproc; int        column;</pre>  |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>column</b><br/>The number of the column of interest. The first column is number 1.</p>   |
| Return value | <i>column</i> 's user-defined datatype or a negative integer if <i>column</i> is not in range.  |
| Usage        | <ul style="list-style-type: none"> <li>• <code>dbccoltype</code> returns the user-defined datatype for a regular result column. For a description of how to add user-defined datatypes to Adaptive Server Enterprise databases, see <code>sp_addtype</code> in the <i>Adaptive Server Enterprise Reference Manual</i>.</li> <li>• <code>dbccoltype</code> is defined as datatype DBINT to accommodate the size of user-defined datatypes. Both DBINT and user-defined datatypes are 32 bits long.</li> <li>• The following code fragment illustrates the use of <code>dbccoltype</code>:</li> </ul> |

```
DBPROCESS      *dbproc;
int            colnum;
int            numcols;

/* Put the command into the command buffer */
dbcmd(dbproc, "select * from mytable");

/*
** Send the command to the Adaptive Server
Enterprise and begin
** execution.
*/
dbsqlxexec(dbproc);

/* Process the command results */
dbresults(dbproc);

/* Examine the user-defined column types */
numcols = dbnumcols(dbproc);
for (colnum = 1; colnum < numcols; colnum++)
{
    printf ("column %d, user-defined type is \
```

```
        %ld.\n", colnum, dbcolutype(dbproc,
        colnum));
    }
```

See also                    dbaltutype, dbcoltype

dbconvert

Description                Convert data from one datatype to another.

Syntax                    DBINT dbconvert(dbproc, srctype, src, srclen,
                              desttype, dest, destlen)

```
DBPROCESS    *dbproc;
int           srctype;
BYTE          *src;
DBINT        srclen;
int           desttype;
BYTE          *dest;
DBINT        destlen;
```

Parameters                dbproc  
                            A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server. In dbconvert, the DBPROCESS is used only to supply any custom null values that the program may have specified using dbsetnull. If *dbproc* is NULL, dbconvert uses the default values for null value data conversions.

srctype  
                            The datatype of the data that is to be converted. This parameter can be any of the server datatypes, as listed below in Table 2-7 on page 110.

src  
                            A pointer to the data which is to be converted. If this pointer is NULL, dbconvert will place an appropriate null value in the destination variable. You can use dbdata to get the server data.

srclen  
                            The length, in bytes, of the data to be converted. If the *srclen* is 0, the source data is assumed to be null and dbconvert will place an appropriate null value in the destination variable. Otherwise, this length is ignored for all datatypes except char, text, binary, and image. For SYBCHAR, SYBBOUNDARY, and SYBSENSITIVITY data, a length of -1 indicates that the string is null-terminated. You can use dbdatlen to get the length of the server data.

**desttype**

The datatype that the source data is to be converted into. This parameter can be any of the server datatypes, as listed below in Table 2-7 on page 110.

**dest**

A pointer to the destination variable that will receive the converted data. If this pointer is NULL, dbconvert will call the user-supplied error handler (if any) and return -1.

**destlen**

The length, in bytes, of the destination variable. *destlen* is ignored for fixed-length datatypes. For a SYBCHAR, SYBBOUNDARY or SYBSENSITIVITY destination, the value of *destlen* must be the total length of the destination buffer space.

Table 2-5 describes special values for *destlen*:

**Table 2-5: Special values for *destlen* (dbconvert)**

| Value of <i>destlen</i> | Applicable to                              | Meaning  |
|-------------------------|--|--|
| -1                      | SYBCHAR,<br>SYBBOUNDARY,<br>SYBSENSITIVITY | There is sufficient space available.<br><br>The string will be trimmed of trailing blanks and given a terminating null.              |
| -2                      | SYBCHAR                                    | There is sufficient space available.<br><br>The string will not be trimmed of trailing blanks, but will be given a terminating null. |

**Return value**

The length of the converted data, in bytes, if the datatype conversion succeeds.

If the conversion fails, dbconvert returns either -1 or FAIL, depending on the cause of the failure. dbconvert returns -1 to indicate a NULL destination pointer or an illegal datatype. dbconvert returns FAIL to indicate other types of failures.

If dbconvert fails, it will first call a user-supplied error handler (if any) and set the global DB-Library error value.

This routine may fail for several reasons: the requested conversion was not available; the conversion resulted in truncation, overflow, or loss of precision in the destination variable; or a syntax error occurred in converting a character string to some numeric type.

**Usage**

- This routine allows the program to convert data from one representation to another. To determine whether a particular conversion is permitted, the program can call dbwillconvert before attempting a conversion.

- dbconvert can convert data stored in any of the server datatypes (although, of course, not all conversions are legal). See Table 2-7 on page 110 for a list of type constants and corresponding program variable types.
- It is an error to use the following datatypes with dbconvert if the library version has not been set (with dbsetversion) to DBVERSION\_100 or higher: SYBNUMERIC, SYBDECIMAL, SYBBOUNDARY, and SYBSENSITIVITY.
- Table 2-8 on page 111 lists the datatype conversions that dbconvert supports. The source datatypes are listed down the leftmost column and the destination datatypes are listed along the top row of the table. (For brevity, the prefix “SYB” has been eliminated from each datatype.) T (“True”) indicates that the conversion is supported; F (“False”) indicates that the conversion is not supported.
- A conversion to or from the datatypes SYBBINARY and SYBIMAGE is a straight bit-copy, except when the conversion involves SYBCHAR or SYBTEXT. When converting SYBCHAR or SYBTEXT data to SYBBINARY or SYBIMAGE, DBCONVERT interprets the SYBCHAR or SYBTEXT string as hexadecimal, whether or not the string contains a leading “0x”. When converting SYBBINARY or SYBIMAGE data to SYBCHAR or SYBTEXT, dbconvert creates a hexadecimal string without a leading “0x”.
- Note that SYBINT2 and SYBINT4 are signed types. When converting these types to character, conversion error can result if the quantity being converted is unsigned and uses the high bit.
- Converting a SYBMONEY, SYBCHAR, or SYBTEXT value to SYBFLT8 may result in some loss of precision. Converting a SYBFLT8 value to SYBCHAR or SYBTEXT may also result in some loss of precision.
- Converting a SYBFLT8 value to SYBMONEY can result in overflow, because the maximum value for SYBMONEY is \$922,337,203,685,477.58.
- If overflow occurs when converting integer or float data to SYBCHAR or SYBTEXT, the first character of the resulting value will contain an asterisk (\*) to indicate the error.
- A conversion to SYBBIT has the following effect: If the value being converted is not 0, the SYBBIT value will be set to 1; if the value is 0, the SYBBIT value will be set to 0.

- `dbconvert` does not offer precision and scale support for numeric and decimal datatypes. When converting to `SYBNUMERIC` or `SYBDECIMAL`, `dbconvert` uses a default precision and scale of 18 and 0, respectively. To specify a different precision and scale, an application can use `dbconvert_ps`.
- `SYBBOUNDARY` and `SYBSENSITIVITY` destinations are always null-terminated.
- In certain cases, it can be useful to convert a datatype to itself. For instance, a conversion of `SYBCHAR` to `SYBCHAR` with a *destlen* of -1 serves as a useful way to append a null terminator to a string, as the example below illustrates.
- Here is a short example that illustrates how to convert server data obtained with `dbdata`:

```
DBCHAR      title[81];
DBCHAR      price[9];

/* Read the query into the command buffer */
dbcmd(dbproc, "select title, price, royalty from \
pubs2..titles");

/* Send the query to Adaptive Server Enterprise */
dbsqlxexec(dbproc);

/* Get ready to process the query results */
dbresults(dbproc);

/* Process each row */
while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
    /*
    ** The first dbconvert() adds a null
    ** terminator to the string.
    */
    dbconvert(dbproc, SYBCHAR, (dbdata(dbproc,1)),
              (dbdatlen(dbproc,1)), SYBCHAR, title,
              (DBINT)-1);

    /*
    ** The second dbconvert() converts money to
    ** string.
    */
    dbconvert(dbproc, SYBMONEY,
              (dbdata(dbproc,2)), (DBINT)-1, SYBCHAR,
              price, (DBINT)-1);
```

```
        if (dbdatlen(dbproc,3) != 0)
            printf ("%s\n $%s  %ld\n", title, price,
                *((DBINT *)dbdata(dbproc,3)));
    }
```

In the dbconvert calls it was not necessary to cast the returns from dbdata, because dbdata returns a BYTE pointer—precisely the datatype dbconvert expects in the third parameter.

- If you are binding data to variables with dbbind rather than accessing the data directly with dbdata, dbbind can perform the conversions itself, making dbconvert unnecessary.
- The sample program *example5.c* illustrates several more types of conversions using dbconvert.
- See Types on page 412 for a list of DB-Library datatypes and the corresponding Adaptive Server Enterprise datatypes. See the *Adaptive Server Enterprise Reference Manual*.

See also

dbaltbind, dbaltbind\_ps, dbbind, dbbind\_ps, dbconvert\_ps, dberrhandle, dbsetnull, dbsetversion, dbwillconvert, Errors on page 389, Types on page 412

## dbconvert\_ps

**Description** Convert data from one datatype to another, with precision and scale support for numeric and decimal datatypes.

**Syntax** DBINT dbconvert\_ps(dbproc, srctype, src, srclen,  
desttype, dest, destlen, typeinfo)

|            |            |
|------------|------------|
| DBPROCESS  | *dbproc;   |
| int        | srctype;   |
| BYTE       | *src;      |
| DBINT      | srclen;    |
| int        | desttype;  |
| BYTE       | *dest;     |
| DBINT      | destlen;   |
| DBTYPEINFO | *typeinfo; |

## Parameters

**dbproc**

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server. In `dbconvert_ps`, the DBPROCESS is used only to supply any custom null values that the program may have specified using `dbsetnull`. If `dbproc` is NULL, `dbconvert_ps` uses the default values for null value data conversions.

**srctype**

The datatype of the data which is to be converted. This parameter can be any of the server datatypes, as listed in Table 2-8 on page 111.

**src**

A pointer to the data that is to be converted. If this pointer is NULL, `dbconvert_ps` will place an appropriate null value in the destination variable. You can use `dbdata` to get the server data.

**srclen**

The length, in bytes, of the data to be converted. If the `srclen` is 0, the source data is assumed to be NULL and `dbconvert_ps` will place an appropriate null value in the destination variable. Otherwise, this length is ignored for all datatypes except char, text, binary, and image. For SYBCHAR data, a length of -1 indicates that the string is null-terminated. You can use `dbdatalen` to get the length of the server data.

**desttype**

The datatype that the source data is to be converted into. This parameter can be any of the server datatypes, as listed in Table 2-8 on page 111.

**dest**

A pointer to the destination variable that will receive the converted data. If this pointer is NULL, `dbconvert_ps` will call the user-supplied error handler (if any) and return -1.

**destlen**

The length, in bytes, of the destination variable. `destlen` is ignored for fixed-length datatypes. For a SYBCHAR, SYBBOUNDARY, or SYBSENSITIVITY destination, the value of `destlen` must be the total length of the destination buffer space.

Table 2-6 describes special values for `destlen`:

**Table 2-6: Special values for destlen (dbconvert\_ps)**

| Value of destlen | Applicable to                              | Meaning  |
|------------------|--|--|
| -1               | SYBCHAR,<br>SYBBOUNDARY,<br>SBYSENSITIVITY | There is sufficient space available.<br><br>The string will be trimmed of trailing blanks and given a terminating null.              |
| -2               | SYBCHAR                                    | There is sufficient space available.<br><br>The string will not be trimmed of trailing blanks, but will be given a terminating null. |

**typeinfo**

A pointer to a DBTYPEINFO structure containing information about the precision and scale of decimal or numeric values. An application sets a DBTYPEINFO structure with values for precision and scale before calling dbconvert\_ps to convert data into DBDECIMAL or DBNUMERIC variables.

If *typeinfo* is NULL:

- If the source value is of type SYBNUMERIC or SYBDECIMAL, dbconvert\_ps picks up precision and scale values from the source. In effect, the source data is copied to the destination space.
- If the source value is not SYBNUMERIC or SYBDECIMAL, dbconvert\_ps uses a default precision of 18 and a default scale of 0.

If *srctype* is not SYBDECIMAL or SYBNUMERIC, *typeinfo* is ignored.

A DBTYPEINFO structure is defined as follows:

```
typedef struct typeinfo {  
    DBINT    precision;  
    DBINT    scale;  
} DBTYPEINFO;
```

Legal values for *precision* are from 1 to 77. Legal values for *scale* are from 0 to 77. *scale* must be less than or equal to *precision*.

**Return value**

The length of the converted data, in bytes, if the datatype conversion succeeds.

If the conversion fails, dbconvert\_ps returns either -1 or FAIL, depending on the cause of the failure. dbconvert\_ps returns -1 to indicate a NULL destination pointer or an illegal datatype. dbconvert\_ps returns FAIL to indicate other types of failures.



If `dbconvert_ps` fails, it will first call a user-supplied error handler (if any) and set the global DB-Library error value.

This routine may fail for several reasons: the requested conversion was not available; the conversion resulted in truncation, overflow, or loss of precision in the destination variable; or a syntax error occurred in converting a character string to some numeric type.

#### Usage

- `dbconvert_ps` is the equivalent of `dbconvert`, except that `dbconvert_ps` provides precision and scale support for numeric and decimal datatypes, which `dbconvert` does not. Calling `dbconvert` is equivalent to calling `dbconvert_ps` with *typeinfo* as `NULL`.
- `dbconvert_ps` allows a program to convert data from one representation to another. To determine whether a particular conversion is permitted, the program can call `dbwillconvert` before attempting a conversion.
- `dbconvert_ps` can convert data stored in any of the server datatypes (but not all conversions are legal—see Table 2-8 on page 111).

Table 2-7 shows type constants for server datatypes and the corresponding program variable types:

**Table 2-7: Type constants and program variable types**

| Server datatype constant | Program variable type |
|--------------------------|-----------------------|
| SYBCHAR                  | DBCHAR                |
| SYBTEXT                  | DBCHAR                |
| SYBBINARY                | DBBINARY              |
| SYBIMAGE                 | DBBINARY              |
| SYBINT1                  | DBTINYINT             |
| SYBINT2                  | DBSMALLINT            |
| SYBINT4                  | DBINT                 |
| SYBFLT8                  | DBFLT8                |
| SYBREAL                  | DBREAL                |
| SYBNUMERIC               | DBNUMERIC             |
| SYBDECIMAL               | DBDECIMAL             |
| SYBBIT                   | DBBIT                 |
| SYBMONEY                 | DBMONEY               |
| SYBMONEY4                | DBMONEY4              |
| SYBDATETIME              | DBDATETIME            |
| SYBDATETIME4             | DBDATETIME4           |
| SYBBOUNDARY              | DBCHAR                |
| SYBSENSITIVITY           | DBCHAR                |

---

**Warning!** It is an error to use the following datatypes with dbconvert\_ps if the library version has not been set (with dbsetversion) to DBVERSION\_100 or higher: SYBNUMERIC, SYBDECIMAL, SYBBOUNDARY, and SYBSENSITIVITY.

---

- Table 2-8 shows the datatype conversions that dbconvert\_ps supports. Source datatypes are listed down the left side, and destination datatypes are listed across the top. (For brevity, the “SYB” datatype prefix is not shown.)

**Table 2-8: Supported datatype conversions**

| From:       | To:  |      |        |       |      |      |      |      |      |         |         |     |       |        |          |           |          |             |
|-------------|------|------|--------|-------|------|------|------|------|------|---------|---------|-----|-------|--------|----------|-----------|----------|-------------|
|             | CHAR | TEXT | BINARY | IMAGE | INT1 | INT2 | INT4 | FLT8 | REAL | NUMERIC | DECIMAL | BIT | MONEY | MONEY4 | DATETIME | DATETIME4 | BOUNDARY | SENSITIVITY |
| CHAR        | •    | •    | •      | •     | •    | •    | •    | •    | •    | •       | •       | •   | •     | •      | •        | •         | •        | •           |
| TEXT        | •    | •    | •      | •     | •    | •    | •    | •    | •    | •       | •       | •   | •     | •      | •        | •         | •        | •           |
| BINARY      | •    | •    | •      | •     | •    | •    | •    | •    | •    | •       | •       | •   | •     | •      |          |           |          |             |
| IMAGE       | •    | •    | •      | •     | •    | •    | •    | •    | •    | •       | •       | •   | •     | •      |          |           |          |             |
| INT1        | •    | •    | •      | •     | •    | •    | •    | •    | •    | •       | •       | •   | •     | •      |          |           |          |             |
| INT2        | •    | •    | •      | •     | •    | •    | •    | •    | •    | •       | •       | •   | •     | •      |          |           |          |             |
| INT4        | •    | •    | •      | •     | •    | •    | •    | •    | •    | •       | •       | •   | •     | •      |          |           |          |             |
| FLT8        | •    | •    | •      | •     | •    | •    | •    | •    | •    | •       | •       | •   | •     | •      |          |           |          |             |
| REAL        | •    | •    | •      | •     | •    | •    | •    | •    | •    | •       | •       | •   | •     | •      |          |           |          |             |
| NUMERIC     | •    | •    | •      | •     | •    | •    | •    | •    | •    | •       | •       | •   | •     | •      |          |           |          |             |
| DECIMAL     | •    | •    | •      | •     | •    | •    | •    | •    | •    | •       | •       | •   | •     | •      |          |           |          |             |
| BIT         | •    | •    | •      | •     | •    | •    | •    | •    | •    | •       | •       | •   | •     | •      |          |           |          |             |
| MONEY       | •    | •    | •      | •     | •    | •    | •    | •    | •    | •       | •       | •   | •     | •      |          |           |          |             |
| MONEY4      | •    | •    | •      | •     | •    | •    | •    | •    | •    | •       | •       | •   | •     | •      |          |           |          |             |
| DATETIME    | •    | •    | •      | •     |      |      |      |      |      |         |         |     |       |        | •        | •         |          |             |
| DATETIME4   | •    | •    | •      | •     |      |      |      |      |      |         |         |     |       |        | •        | •         |          |             |
| BOUNDARY    | •    | •    |        |       |      |      |      |      |      |         |         |     |       |        |          |           | •        |             |
| SENSITIVITY | •    | •    |        |       |      |      |      |      |      |         |         |     |       |        |          |           |          | •           |

- A conversion to or from the datatypes SYBBINARY and SYBIMAGE is a straight bit-copy, except when the conversion involves SYBCHAR or SYBTEXT. When converting SYBCHAR or SYBTEXT data to SYBBINARY or SYBIMAGE, dbconvert\_ps interprets the SYBCHAR or SYBTEXT string as hexadecimal, whether or not the string contains a leading "0x." When converting SYBBINARY or SYBIMAGE data to SYBCHAR or SYBTEXT, dbconvert\_ps creates a hexadecimal string without a leading "0x."
- Note that SYBINT2 and SYBINT4 are signed types. When converting these types to character, conversion error can result if the quantity being converted is unsigned and uses the high bit.

- Converting a SYBMONEY, SYBCHAR, or SYBTEXT value to SYBFLT8 may result in some loss of precision. Converting a SYBFLT8 value to SYBCHAR or SYBTEXT may also result in some loss of precision.
- Converting a SYBFLT8 value to SYBMONEY can result in overflow, because the maximum value for SYBMONEY is \$922,337,203,685,477.58.
- If overflow occurs when converting integer or float data to SYBCHAR or SYBTEXT, the first character of the resulting value will contain an asterisk (\*) to indicate the error.
- A conversion to SYBBIT has the following effect: If the value being converted is not 0, the SYBBIT value will be set to 1; if the value is 0, the SYBBIT value will be set to 0.
- SYBBOUNDARY and SYBSENSITIVITY destinations are always null-terminated.
- In certain cases, it can be useful to convert a datatype to itself. For instance, a conversion of SYBCHAR to SYBCHAR with a *destlen* of -1 serves as a useful way to append a null terminator to a string.
- If you are binding data to variables with `dbbind` or `dbbind_ps` rather than accessing the data directly with `dbdata`, `dbbind` can perform the conversions itself, making `dbconvert_ps` unnecessary.
- The sample program *example5.c* illustrates several more types of conversions using `dbconvert_ps`.
- See Types on page 412 for a list of DB-Library datatypes and the corresponding Adaptive Server Enterprise datatypes. See the *Adaptive Server Enterprise Reference Manual*.

See also

`dbaltbind`, `dbaltbind_ps`, `dbbind`, `dbbind_ps`, `dbconvert`, `dberrhandle`, `dbsetnull`, `dbsetversion`, `dbwillconvert`, Errors on page 389, Types on page 412

## DBCOUNT

Description

Returns the number of rows affected by a Transact-SQL command.

Syntax

DBINT DBCOUNT(dbproc)

DBPROCESS \*dbproc;

|              |  |
|--------------|--|
| Parameters   | <p><code>dbproc</code></p> <p>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>   |
| Return value | <p>The number of rows affected by the command, or -1. <code>DBCOUNT</code> will return -1 if any of the following are true:</p> <ul style="list-style-type: none"><li>• The Transact-SQL command fails for any reason, such as a syntax error.</li><li>• The command is one that never affects rows, such as a print command.</li><li>• The command executes a stored procedure that does not execute any select statements.</li><li>• The <code>DBNOCOUNT</code> option is on.</li></ul>  |
| Usage        | <ul style="list-style-type: none"><li>• Once the results of a command have been processed, you can call <code>DBCOUNT</code> to find out how many rows were affected by the command. For example, if a select command was sent to the server and you have read all the rows by calling <code>dbnextrow</code> until it returned <code>NO_MORE_ROWS</code>, you can call this macro to find out how many rows were retrieved.</li><li>• If the current command is one that does not return rows, (for example, a delete), you can call <code>DBCOUNT</code> immediately after <code>dbresults</code>.</li><li>• If the command is one that executes a stored procedure, for example an <code>exec</code> or remote procedure call, <code>DBCOUNT</code> returns the number of rows returned by the latest select statement executed by the stored procedure, or -1 if the stored procedure does not execute any select statements. Note that a stored procedure that contains no select statements may execute a select by calling another stored procedure that does contain a select.</li></ul> |
| See also     | <code>dbnextrow</code> , <code>dbresults</code> , Options on page 407  |

## DBCURCMD

|             |  |
|-------------|--|
| Description | Return the number of the current command.            |
| Syntax      | <pre>int DBCURCMD(dbproc)  DBPROCESS  *dbproc;</pre> |

|              |  |
|--------------|--|
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>  |
| Return value | The number of the current command.   |
| Usage        | <ul style="list-style-type: none"> <li>• This macro returns the number of the command whose results are currently being processed.</li> <li>• The first command in a batch is number 1. The command number is incremented every time dbresults returns SUCCEED or FAIL. (Unsuccessful commands are counted.) The command number is reset by each call to dbsqlxexec or dbsqlsend.</li> </ul> |
| See also     | DBCMDROW, DBMORECMDS, DBROWS   |

## DBCURROW

|              |  |
|--------------|--|
| Description  | Return the number of the row currently being read.   |
| Syntax       | <p>DBINT DBCURROW(dbproc)</p> <p>DBPROCESS *dbproc;</p>  |
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>  |
| Return value | The number of the current row. This routine returns 0 if no rows have been processed yet.  |
| Usage        | <ul style="list-style-type: none"> <li>• This macro returns the number of the row currently being read. Rows are counted from the first row returned from the server, whose number is 1. DBCURROW counts both regular and compute rows.</li> <li>• The row number is reset to 0 by each new call to dbresults.</li> <li>• The row number grows by one every time dbnextrow returns REG_ROW or a <i>computeid</i>.</li> </ul> |

- When row buffering is used, the row number *does not* represent the position in the row buffer. Rather, it represents the current row's position in the rows returned by the server. See the reference pages for `dbgetrow` and `dbsetrow`.

See also

`dbclrbuf`, `DBFIRSTROW`, `dbgetrow`, `DBLASTROW`, `dbnextrow`, `dbsetopt`, Options on page 407

## dbcursor

Description

Insert, update, delete, lock, or refresh a particular row in the fetch buffer.

Syntax

`RETCODE dbcursor(hc, optype, bufno, table, values)`

```
DBCURSOR    *hc;
DBINT       optype;
DBINT       bufno;
BYTE        *table;
BYTE        *values
```

Parameters

`hc`

Cursor handle previously returned by `dbcursoropen`.

`optype`

Type of operation to perform. Table 2-9 lists the operation types.

**Table 2-9: Values for `optype` (`dbcursor`)**

| Symbolic value           | Operation   |
|--------------------------|---|
| <code>CRS_UPDATE</code>  | Updates data.   |
| <code>CRS_DELETE</code>  | Deletes data.   |
| <code>CRS_INSERT</code>  | Inserts data.   |
| <code>CRS_REFRESH</code> | Fetches another row in the buffer.  |
| <code>CRS_LOCKCC</code>  | Fetches another row and locks it. The row is actually locked only if inside a transaction block. The lock is released when the application commits or ends the transaction. |

`bufno`

Row number in the fetch buffer to which the operation applies. The specified buffer must contain a valid row. If the value of `bufno` is 0, a `CRS_REFRESH` operation applies to all rows in the buffer. In an insert or update operation where no values parameter is given, the values are read from the bound variables array in the corresponding `bufno` value. The number of the first row in the buffer is 1.

**table**

The table to be inserted, updated, deleted, or locked if the cursor declaration contains more than one table. If there is only one table, this parameter is not required.

**values**

String values to be updated and/or inserted. Use this parameter only with update and insert to specify the new column values (that is, *Quantity* = *Quantity* + 1). In most cases, you can set this parameter to NULL and the new values for each column are taken from the fetch buffer (the program variable specified by `dbcursorbind`). If the select statement includes a computation (that is, `select 5*5...`) and a function (for example, `select getdate()`, `convert()`, and so on), then updating through the buffer array will surely not work.

There are four possible formats for this parameter: two for updating and two for inserting. The chosen format must match the *optype* (update or insert). Both contain a full and an abbreviated format. The full format is a complete SQL statement (update or insert) without a where clause. The abbreviated format is just the set clause (update) or just the values clause (insert). When the full format is used, the value specified for *tablename* overrides the *table* parameter of `dbcursor`. Because a where clause is added automatically, do not include one.

**Return value**

SUCCEED or FAIL.

This function can fail for the following reasons:

- Cursor is opened as read only, no updates allowed.
- Server or connection failure or timeout.
- No permission to update or change the database.
- A trigger in the database caused the lock or update/insert operation to fail.
- Optimistic concurrency control.

**Usage**

- If a column used as a unique index column is updated or changed, the corresponding row appears to be missing the next time it is fetched.
- See Appendix A, “Cursors”.

**See also**

`dbcursorbind`, `dbcursorclose`, `dbcursorcolinfo`, `dbcursorfetch`, `dbcursorinfo`, `dbcursoropen`



## dbcursorbind

|             |  |
|-------------|--|
| Description | Register the binding information on the cursor columns.  |
| Syntax      | <pre> RETCODE dbcursorbind(hc, col, vartype, varlen,                      poutlen, pvaraddr, typeinfo)  DBCURSOR *hc; int       col; int       vartype; DBINT     varlen; DBINT     *poutlen; BYTE     *pvaraddr; DBTYPEINFO *typeinfo; </pre>   |
| Parameters  | <p><b>hc</b><br/>Cursor handle created by <code>dbcursoropen</code>.</p> <p><b>col</b><br/>Number of the column to be bound to a program variable.</p> <p><b>vartype</b><br/>Binding type, which uses the same datatypes as the <i>vartype</i> parameter for <code>dbbind</code> and is bound by the same conversion rules. If this value is set to <code>NOBIND</code> for any column, the data is not bound. Instead, a pointer to the data is returned to the address in the corresponding <i>pvaraddr</i> entry for every row, and the length of the data is returned to the corresponding <i>varlen</i> array entry. This feature lets the application access the cursor data as it does with <code>dbdata</code> and <code>dbdatalen</code>.</p> <p><b>varlen</b><br/>Maximum length of variable-length datatype, such as <code>CHARBIND</code>, <code>VARYCHARBIND</code>, <code>BINARYBIND</code>, <code>STRINGBIND</code>, <code>NTBSTRINGBIND</code>, and <code>VARYBINBIND</code>. This parameter is ignored for fixed-length datatypes, such as <code>INTBIND</code>, <code>FLT8BIND</code>, <code>MONEYBIND</code>, <code>BITBIND</code>, <code>SMALLBIND</code>, and so on. It is also ignored if the <i>vartype</i> is <code>NOBIND</code>.</p> |

**poutlen**

Pointer to an array of DBINT integers where the actual length of the column's data is returned for each row. If *poutlen* is set to NULL, the lengths are not returned. The array size must be large enough to hold one DBINT variable for every row to be fetched at a time (as indicated by the *nrows* parameter in *dbcursoropen*).

When using *dbcursor* to update or insert with values from bound program variables, you can specify a null value by setting the corresponding *poutlen* to zero before calling *dbcursor*. Nonzero values are ignored except when NOBIND or one of the variable-length datatypes such as VARYCHARBIND or VARYBINBIND has been specified. In that case *poutlen* must contain the actual item length. If STRINGBIND or NTBSTRINGBIND has been specified, any non-zero value for *poutlen* is ignored, and the length of the string is determined by scanning for the null terminator.

**pvaraddr**

Pointer to the program variable to which the data is copied. If *vartype* is NOBIND, *pvaraddr* is assumed to point to an array of pointers—to the address of the actual data fetched by *dbcursorfetch*. This array's length must equal the value of *nrows* in *dbcursoropen*. If the cursor was opened with *nrows* > 1, *pvaraddr* is assumed to point to an array of *nrows* elements. Calling *dbcursorbind* with *pvaraddr* set to NULL breaks the existing binding.

**typeinfo**

A pointer to a DBTYPEINFO structure containing information about the precision and scale of decimal or numeric values. If *vartype* is not DECIMALBIND or NUMERICBIND, *typeinfo* is ignored.

To bind to DBNUMERIC or DBDECIMAL variables, an application initializes a DBTYPEINFO structure with values for precision and scale, then calls *dbcursorbind* with *vartype* as DECIMALBIND or NUMERICBIND.

If *typeinfo* is NULL and *vartype* is DECIMALBIND or NUMERICBIND:

- If the result column is of type numeric or decimal, *dbcursorbind* picks up precision and scale values from the result column.
- If the result column is not numeric or decimal, *dbcursorbind* uses a default precision of 18 and a default scale of 0.

A DBTYPEINFO structure is defined as follows:

```
typedef struct typeinfo {
```

```

        DBINTprecision;
        DBINTscale;
    } DBTYPEINFO;

```

Legal values for *precision* are from 1 to 77. Legal values for *scale* are from 0 to 77. *scale* must be less than or equal to *precision*.

|              |   |
|--------------|---|
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"> <li>• If <code>dbcursorbind</code> is called more than once for any column, only the last call is effective.</li> <li>• This function works almost the same as <code>dbbind</code> without cursors.</li> <li>• See Appendix A, “Cursors”.</li> </ul> |
| See also     | <code>dbcursor</code> , <code>dbcursorclose</code> , <code>dbcursorcolinfo</code> , <code>dbcursorfetch</code> , <code>dbcursorinfo</code> , <code>dbcursoropen</code>  |

## dbcursorclose

|              |   |
|--------------|---|
| Description  | Close the cursor associated with the given handle and release all the data belonging to it.   |
| Syntax       | <pre>void dbcursorclose(hc)</pre> <pre>DBCURSOR    *hc;</pre>   |
| Parameters   | <p><code>hc</code></p> <p>Cursor handle created by <code>dbcursoropen</code>.</p>   |
| Return value | None.   |
| Usage        | <ul style="list-style-type: none"> <li>• Closing a DBPROCESS connection with <code>dbcursorclose</code> automatically closes all the cursors associated with it. After issuing <code>dbcursorclose</code>, the cursor handle should not be used.</li> <li>• See Appendix A, “Cursors”.</li> </ul> |
| See also     | <code>dbcursor</code> , <code>dbcursorbind</code> , <code>dbcursorcolinfo</code> , <code>dbcursorfetch</code> , <code>dbcursorinfo</code> , <code>dbcursoropen</code>   |

## dbcursorcolinfo

|              |   |
|--------------|---|
| Description  | Return column information for the specified column number in the open cursor.   |
| Syntax       | <pre>RETCODE dbcursorcolinfo(hcursor, column, colname,                         coltype, collen, usertype)  DBCUSOR    *hcursor DBINT      column; DBCHAR     *colname; DBINT      *coltype; DBINT      *collen; DBINT      *usertype;</pre>   |
| Parameters   | <p><b>hcursor</b><br/>Cursor handle created by <code>dbcursoropen</code>.</p> <p><b>column</b><br/>Column number for which information is to be returned.</p> <p><b>colname</b><br/>Location where the name of the column is returned. The user should allocate space large enough to accommodate the column name.</p> <p><b>coltype</b><br/>Location where the column's datatype is returned.</p> <p><b>collen</b><br/>Location where the column's maximum length is returned.</p> <p><b>usertype</b><br/>Location where the column's user-defined datatype is returned.</p> |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"> <li>Any of the parameters <i>colname</i>, <i>coltype</i>, <i>collen</i>, or <i>usertype</i> can be set to NULL, in which case the information for that variable is not returned.</li> <li>See Appendix A, "Cursors"</li> </ul>   |
| See also     | <code>dbcursor</code> , <code>dbcursorbind</code> , <code>dbcursorclose</code> , <code>dbcursorfetch</code> , <code>dbcursorinfo</code> , <code>dbcursoropen</code>   |

## dbcursorfetch

|             |  |
|-------------|--|
| Description | Fetch a block of rows into the program variables declared by the user in <code>dbcursorbind</code> .   |
| Syntax      | <pre>RETCODE dbcursorfetch(hc, fetchtype, rownum)</pre><br><pre>DBCURSOR    *hc;<br/>DBINT        fetchtype;<br/>DBINT        rownum;</pre>  |
| Parameters  | <p><code>hc</code><br/>Cursor handle created by <code>dbcursoropen</code>.</p> <p><code>fetchtype</code><br/>Type of fetch chosen. The <code>scroll</code> option in <code>dbcursoropen</code> determines which of these values are legal. Table 2-10 lists the various fetch types.</p> |

**Table 2-10: Values for fetchtype (dbcursorfetch)**

| Symbolic value | Meaning  | Comment  |
|----------------|--|--|
| FETCH_FIRST    | Fetch the first block of rows.   | Although available for all cursor types, this option is especially useful for returning to the beginning of a keyset when you have selected a forward-only scrolling cursor.   |
| FETCH_NEXT     | Fetch the next block of rows.  | If the result set exceeds the specified keyset size and if FETCH_RANDOM and/or FETCH_RELATIVE have been issued, a FETCH_NEXT can span a keyset boundary. In this case, the fetch that spans a keyset boundary returns a partial buffer, and the next fetch shifts down the keyset and returns the next full set of rows. |
| FETCH_PREV     | Fetch the previous block of rows.  | This option is unavailable with forward-only scrolling cursors. If <i>rownum</i> falls within the keyset, the range of rows must stay within the keyset because only the rows within the keyset are returned. This option does not change the keyset to the previous <i>rownum</i> rows in the result set.               |
| FETCH_RANDOM   | Fetch a block of rows, starting from the specified row number within the keyset.   | This option is valid only within the keyset. The buffer is only partially filled when the range spans the keyset boundary.   |
| FETCH_RELATIVE | Fetch a block of rows, relative to the number of rows indicated in the last fetch. | This option jumps <i>rownum</i> rows from the first row of the last fetch and starts fetching from there. The rows must remain within the keyset. The buffer is only partially filled when the range spans the keyset boundary.  |
| FETCH_LAST     | Fetch the last block of rows.  | This value is available only with totally keyset-driven cursors.   |

*rownum*

The specified row for the buffer to start filling. Use this parameter only with a *fetchtype* of FETCH\_RANDOM or -FETCH\_RELATIVE.

Return value

SUCCEED or FAIL.

If the status array contains a status row for every row fetched, SUCCEED is returned. FAIL is returned if at least one of the following is true.

- FETCH\_RANDOM and FETCH\_RELATIVE require a keyset driven cursor.
- Forward-only scrolling can use only FETCH\_FIRST and FETCH\_NEXT.
- The server or a connection fails or takes a timeout.
- The client is out of memory.
- The FETCH\_LAST option requires a fully keyset-driven cursor.
- Specify the size of the fetch buffer in dbcursoropen. dbcursorfetch fills the array passed as dbcursoropen's *pstatus* parameter with status codes for the fetched rows. See the reference page for dbcursoropen for these codes.
- Program variables must first be registered, using dbcursorbind. Then the data can be transferred into the DB-Library buffers. The bound variables must, therefore, be arrays large enough to hold the specified number of rows. The status array contains status code for every row and contains flags for missing rows.
- When the range of rows specified by FETCH\_NEXT, FETCH\_RANDOM, or FETCH\_RELATIVE spans a keyset boundary, only the rows remaining in the keyset are returned. In this case, the buffer is only partially filled, and the FETCH\_ENDOFKEYSET flag is set as the status of the last row. The following FETCH\_NEXT shifts the keyset down.
- See Appendix A, "Cursors"

#### Usage

#### See also

dbcursor, dbcursorbind, dbcursorclose, dbcursorcolinfo, dbcursorinfo, dbcursoropen

## dbcursorinfo

#### Description

Return the number of columns and the number of rows in the keyset if the keyset hit the end of the result set.

#### Syntax

```
RETCODE dbcursorinfo(hcursor, ncols, nrows);
```

```
DBCURSOR *hcursor;
DBINT *ncols;
DBINT *nrows;
```

|              |   |
|--------------|---|
| Parameters   | <p><code>hcursor</code><br/>Cursor handle created by <code>dbcursoropen</code>.</p> <p><code>ncols</code><br/>Location where the number of columns in the cursor is returned.</p> <p><code>nrows</code><br/>Location where the number of rows in the keyset is returned.</p>  |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"><li>For fully keyset-driven cursors, the <i>nrows</i> parameter contains the number of rows in the keyset. For mixed or dynamic cursors, <i>nrows</i> is always set to -1, unless the keyset is the last one in the result set. In that case, the number of rows in the keyset is returned. This helps the programmer find out when the keyset has hit the end of the result set.</li><li>See Appendix A, “Cursors”</li></ul> |
| See also     | <code>dbcursor</code> , <code>dbcursorbind</code> , <code>dbcursorclose</code> , <code>dbcursorcolinfo</code> , <code>dbcursorfetch</code> , <code>dbcursoropen</code>  |

## dbcursoropen

|             |  |
|-------------|--|
| Description | Open a cursor and specify the scroll option, concurrency option, and the size of the fetch buffer (the number of rows retrieved with a single fetch).  |
| Syntax      | <pre>DBCURSOR *dbcursoropen(dbproc, stmt, scrollopt,<br/>                        concuropt, nrows, pstatus)</pre><br><pre>DBPROCESS  *dbproc;<br/>BYTE       *stmt;<br/>SHORT      scrollopt;<br/>SHORT      concuropt;<br/>USHORT     nrows;<br/>DBINT      *pstatus</pre>  |
| Parameters  | <p><code>dbproc</code><br/>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><code>stmt</code><br/>The select statement that defines a cursor.</p> |



**scrollopt**

Indicator of the desired scrolling technique.

*Keyset driven* fixes membership in the result set and order at cursor open time.

*Dynamic* determines membership in the result set and order at fetch time.

Table 2-11 lists the possible values for *scrollopt*.

**Table 2-11: Values for *scrollopt* (*dbcursoropen*)**

| Symbolic value | Meaning  |
|----------------|--|
| CUR_FORWARD    | Forward scrolling only.  |
| CUR_KEYSET     | Keyset driven. A copy of the keyset for the result table is kept locally. Number of rows in result table must be less than or equal to 1000. |
| CUR_DYNAMIC    | Fully dynamic.   |
| int <i>n</i>   | Keyset-driven cursor within ( <i>n*rows</i> ) blocks, but fully dynamic outside the keyset.  |

**concurop**

Definition of concurrency control. Table 2-12 lists the possible values for *concurop*:

**Table 2-12: Values for *concurop* (*dbcursoropen*)**

| Symbolic value | Meaning  | Explanation  |
|----------------|--|--|
| CUR_READONLY   | Read-only cursor.  | The data cannot be modified.   |
| CUR_LOCKCC     | Intent to update locking.                                  | All data, if inside a transaction block, is locked out as it is fetched through <i>dbcursorfetch</i> .   |
| CUR_OPTCC      | Optimistic concurrency control, based on timestamp values. | In a given row, modifications to the data succeed only if the row has not been updated since the last fetch. Changes are detected through timestamps or by comparing all non-text, non-image values in a selected table row. |
| CUR_OPTCCVAL   | Optimistic concurrency based on values.                    | Same as CUR_OPTCC except changes are detected by comparing the values in all selected columns.   |

**nrows**

Number of rows in the fetch buffer (the width of the cursor). For mixed cursors the keyset capacity in rows is determined by this number multiplied by the value of the *scrollopt* parameter.

**pstatus**

Pointer to the array of row status indicators. The status of every row copied into the fetch buffer is returned to this array. The array must be large enough to hold one DBINT integer for every row in the buffer to be fetched. During the *dbcursorfetch* call, as the rows are filled into the bound variable, the corresponding status is filled with status information. *dbcursorfetch* fills in the status by setting bits in the status value. The application can use the bitmask values shown in Table 2-13 to inspect the status value:

**Table 2-13: Bitmask values for *pstatus* (*dbcursoropen*)**

| Symbolic value   | Meaning  |
|------------------|--|
| FTC_SUCCEED      | The row was successfully copied. If this flag is not set, the row was not fetched. |
| FTC_MISSING      | The row is missing.  |
| FTC_ENDOFKEYSET  | The end of the keyset. The remaining rows in the bind arrays are not used.         |
| FTC_ENDOFRESULTS | The end of the result set. The remaining rows are not used.                        |

**Return value**

If *dbcursoropen* succeeds, a handle to the cursor is returned. The cursor handle is required in calls to subsequent cursor functions.

If *dbcursoropen* fails, NULL is returned. Several errors, such as the following, can cause the cursor open to fail:

- Not enough memory in the system. Reduce the number of rows in the keyset, use dynamic scrolling, or reduce the number of rows to be fetched at a time.
- The CUR\_KEYSET option is used for the *scrollopt* parameter, and there are more than 1000 rows in the result set. Use dynamic scrolling if the select statement can return more than 1000 rows.
- A unique row identifier could not be found.

**Usage**

- This function prepares internal DB-Library data structures based on the contents of the select statement and the values of *scrollopt*, *concuropt*, and *nrows*. *dbcursoropen* queries the server for information on unique qualifiers (row keys) for the rows in the cursor result set. If the cursor is keyset-driven, *dbcursoropen* queries the server and fetches row keys to build a keyset for the cursor's rows.
- The cursor definition cannot contain stored procedures or multiple Transact-SQL statements.
- For *dbcursor* to succeed, every table in the select statement must have a unique index. The Transact-SQL statements for browse, select into, compute, union, or compute by are not allowed in the cursor statement. Only fully keyset-driven cursors can have order, having, or group by phrases.
- When the select statement given as *stmt* refers to temporary tables, the current database must be *tempdb*. This restriction applies even if the temporary table was created in another database.

- Multiple cursors (as many as the system's memory allows) can be opened within the same *dbproc* connection. There should be no commands waiting to be executed or results pending in the DBPROCESS connection when cursor functions are called.
- See Appendix A, "Cursors"

See also

dbcursor, dbcursorbind, dbcursorclos, dbcursocolinfo, dbcursorfet, dbcursorinfo, dbcursoropen

## dbdata

Description

Return a pointer to the data in a regular result column.

Syntax

BYTE \*dbdata(dbproc, column)

DBPROCESS \*dbproc;  
int column;

Parameters

dbproc

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

column

The number of the column of interest. The first column is number 1.

Return value

A BYTE pointer to the data for the particular column of interest. Be sure to cast this pointer into the proper type. A NULL BYTE pointer is returned if there is no such column or if the data has a null value. To make sure that the data is really a null value, you should always check for a return of 0 from dbdatlen.

Usage

- This routine returns a pointer to the data in a regular (that is, non-compute) result column. The data is not null-terminated. You can use dbdatlen to get the length of the data.
- Here is a small program fragment that uses dbdata:

```
DBPROCESS      *dbproc;  
DBINT          row_number = 0;  
DBINT          object_id;  
  
/* Put the command into the command buffer */  
dbcmd(dbproc, "select id from sysobjects");  
/*
```

```

    ** Send the command to Adaptive Server Enterprise
    and begin
    ** execution
    */
    dbsqlxexec(dbproc);
    /* Process the command results */
    dbresults(dbproc);
    /* Examine the data in each row */
    while (dbnextrow(dbproc) != NO_MORE_ROWS)
    {
        row_number++;
        object_id = *((DBINT *)dbdata(dbproc, 1));
        printf("row %ld, object id is %ld.\n",
            row_number, object_id);
    }

```

- Do not add a null terminator to string data until you have copied it from the DBPROCESS with a routine such as `strncpy`. For example:

```

char    objname[40];
...
strncpy(objname, (char *)dbdata(dbproc, 2),
    (int)dbdatlen(dbproc, 2));
objname[dbdatlen(dbproc, 2)] = '\0';

```

- The function `dbbind` will automatically bind result data to your program variables. It does a copy of the data, but is often easier to use than `dbdata`. Furthermore, it includes a convenient type-conversion capability. By means of this capability, the application can, among other things, easily add a null terminator to a result string or convert money and datetime data to printable strings.

See also

`dbbind`, `dbcollen`, `dbcolname`, `dbcoltype`, `dbdatlen`, `dbnumcols`

## dbdate4cmp

**Description** Compare two DBDATETIME4 values.

**Syntax** `int dbdate4cmp(dbproc, d1, d2)`

```

DBPROCESS    *dbproc;
DBDATETIME4  *d1;
DBDATETIME4  *d2;

```

|              |   |
|--------------|---|
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL.</p> <p><b>d1</b><br/>A pointer to a DBDATETIME4 value.</p> <p><b>d2</b><br/>A pointer to a DBDATETIME4 value.</p> |
| Return value | <p>If <math>d1 = d2</math>, dbdate4cmp returns 0.</p> <p>If <math>d1 &lt; d2</math>, dbdate4cmp returns -1.</p> <p>If <math>d1 &gt; d2</math>, dbdate4cmp returns 1.</p>  |
| Usage        | <ul style="list-style-type: none"><li>• dbdate4cmp compares two DBDATETIME4 values.</li><li>• The range of legal DBDATETIME4 values is from January 1, 1900 to June 6, 2079. DBDATETIME4 values have a precision of one minute.</li></ul>   |
| See also     | dbdatecmp, dbmnycmp, dbmny4cmp  |

## dbdate4zero

|              |  |
|--------------|--|
| Description  | Initialize a DBDATETIME4 variable to Jan 1, 1900 12:00AM.  |
| Syntax       | <pre>RETCODE dbdate4zero(dbproc, dateptr)</pre> <pre>DBPROCESS  *dbproc;<br/>DBDATETIME4 *dateptr;</pre>   |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL.</p> <p><b>dateptr</b><br/>A pointer to the DBDATETIME4 variable to initialize.</p> |
| Return value | SUCCEED or FAIL.   |

`dbdate4zero` returns FAIL if *dateptr* is NULL.

**Usage**

- `dbdate4zero` initializes a DBDATETIME4 variable to Jan 1, 1900 12:00AM.
- The range of legal DBDATETIME4 values is from January 1, 1900 to June 6, 2079. DBDATETIME4 values have a precision of one minute.

**See also**

`dbdatezero`

## dbdatechar

**Description**

Convert an integer component of a DBDATETIME value into character format.

**Syntax**

RETCODE `dbdatechar`(dbproc, charbuf, datepart, value)

```
DBPROCESS  *dbproc;  
char        *charbuf;  
int         datepart;  
int         value;
```

**Parameters**

**dbproc**

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

**charbuf**

A pointer to the character buffer that will contain the null-terminated character representation of *value*.

**datepart**

A symbolic constant describing *value*'s type. Table 2-14 lists the date parts, the date part symbols recognized by DB-Library, and the expected values. Note that the names of the months and the days in this table are those for English.

**Table 2-14: Date parts and their character representations (dbdatechar)**

| Date part   | Symbol    | Character representation of value |
|-------------|-----------|-----------------------------------|
| year        | DBDATE_YY | 1753 – 9999                       |
| quarter     | DBDATE_QQ | 1 – 4                             |
| month       | DBDATE_MM | January – December                |
| day of year | DBDATE_DY | 1 – 366                           |
| day         | DBDATE_DD | 1 – 31                            |
| week        | DBDATE_WK | 1 – 54 (for leap years)           |
| weekday     | DBDATE_DW | Monday – Sunday                   |
| hour        | DBDATE_HH | 0 – 23                            |
| minute      | DBDATE_MI | 0 – 59                            |
| second      | DBDATE_SS | 0 – 59                            |
| millisecond | DBDATE_MS | 0 – 999                           |

value  
The numeric value to be converted.

Return value  
SUCCEED or FAIL.

- Usage
- dbdatechar converts integer datetime components to character format. For example, dbdatechar associates the month component “3” with its associated character string: “March” if English is used, “mars” if French is used, and so on.
  - The language of the associated character string is determined by the *dbproc*.
  - dbdatechar is often useful in conjunction with dbdatecrack.

See also  
dbconvert, dbdata, dbdatetime, dbdatecrack

# dbdatecmp

Description  
Compare two DBDATETIME values.

Syntax  
int dbdatecmp(dbproc, d1, d2)

```
DBPROCESS  *dbproc;  
DBDATETIME *d1;  
DBDATETIME *d2;
```



|              |   |
|--------------|---|
| Parameters   | <b>dbproc</b><br>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.<br><br>This parameter may be NULL. |
|              | <b>d1</b><br>A pointer to a DBDATETIME value.   |
|              | <b>d2</b><br>A pointer to a DBDATETIME value.   |
| Return value | If $d1 = d2$ , dbdatecmp returns 0.   |
|              | If $d1 < d2$ , dbdatecmp returns -1.  |
|              | If $d1 > d2$ , dbdatecmp returns 1.   |
| Usage        | <ul style="list-style-type: none"> <li>• dbdatecmp compares two DBDATETIME values.</li> <li>• The range of legal DBDATETIME values is from January 1, 1753 to December 31, 9999. DBDATETIME values have a precision of 1/300th of a second (3.33 milliseconds).</li> </ul>                    |
| See also     | dbdate4cmp, dbmnycmp, dbmny4cmp   |

## dbdatecrack

|             |  |
|-------------|--|
| Description | Convert a machine-readable DBDATETIME value into user-accessible format.   |
| Syntax      | <pre>RETCODE dbdatecrack(dbproc, dateinfo, datetime)</pre> <pre>DBPROCESS  *dbproc; DBDATEREC  *dateinfo; DBDATETIME *datetime;</pre>  |
| Parameters  | <b>dbproc</b><br>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server. |
|             | <b>dateinfo</b><br>A pointer to a DBDATEREC structure to contain the parts of <i>datetime</i> . DBDATEREC is defined as follows:   |

```
typedef struct dbdaterec
{
    long    dateyear;      /* 1900 to the future */
    long    datemonth;     /* 0 - 11 */
    long    datedmonth;    /* 1 - 31 */
    long    datedyear;     /* 1 - 366 */
    long    datedweek;     /* 0 - 6 */
    long    datehour;      /* 0 - 23 */
    long    dateminute;    /* 0 - 59 */
    long    datessecond;   /* 0 - 59 */
    long    datemsecond;   /* 0 - 997 */
    long    datetzone;     /* 0 - 127 */
} DBDATEREC;
```

Month and day names depend on the national language of the DBPROCESS. To retrieve these, use dbdatetime or dbdayname plus dbmonthname.

---

**Note** The *dateinfo->datetzone* field is not set by dbdatecrack.

---

datetime

A pointer to the DBDATETIME value of interest.

Return value

SUCCEED or FAIL.

Usage

- dbdatecrack converts a DBDATETIME value into its integer components and places those components into a DBDATEREC structure.
- DBDATETIME structures store date and time values in an internal format. For example, a time value is stored as the number of 300th's of a second since midnight, and a date value is stored as the number of days since January 1, 1900. dbdatecrack converts the internal value to something more usable by an application program.
- The integer date parts placed in the DBDATEREC structure may be converted to character strings using dbdatechar.
- Calling dbdatecrack to convert an internal format datetime value is equivalent to calling dbdatepart many times.
- The following code fragment illustrates the use of dbdatecrack:

```
dbcmd(dbproc, "select name, crdate from \
            master..sysdatabases");
dbsqlxexec(dbproc);
dbresults(dbproc);
while (dbnextrow(dbproc) != NO_MORE_ROWS)
```

```

{
    /*
    ** Print the database name and its date info
    */
    dbconvert(dbproc, dbcoltype(dbproc, 2),
              dbdata(dbproc, 2), dbdatlen(dbproc, 2),
              SYBCHAR, datestring, -1);
    printf("%s: %s\n", (char *)
           (dbdata(dbproc, 1)), datestring);
    /*
    ** Break up the creation date into its
    ** constituent parts.
    */
    dbdatecrack(dbproc, &dateinfo,
                (DBDATETIME *) (dbdata(dbproc, 2)));

    /* Print the parts of the creation date */
    printf("\tYear = %d.\n", dateinfo.dateyear);
    printf("\tMonth = %d.\n", dateinfo.datemonth);
    printf("\tDay of month = %d.\n",
           dateinfo.datedmonth);
    printf("\tDay of year = %d.\n",
           dateinfo.datedyear);
    printf("\tDay of week = %d.\n",
           dateinfo.datedweek);
    printf("\tHour = %d.\n", dateinfo.datehour);
    printf("\tMinute = %d.\n",
           dateinfo.dateminute);
    printf("\tSecond = %d.\n",
           dateinfo.datesecond);
    printf("\tMillisecond = %d.\n",
           dateinfo.datemsecond);
}

```

See also `dbconvert`, `dbdata`, `dbdatechar`, `dbdatename`, `dbdatepart`

## dbdatename

**Description** Convert the specified component of a DBDATETIME structure into its corresponding character string.

**Syntax** `int dbdatename(dbproc, charbuf, datepart, datetime)`

DBPROCESS \*dbproc;

char               \*charbuf;  
int                datepart;  
DBDATETIME       \*datetime;

**Parameters****dbproc**

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

**charbuf**

A pointer to a character buffer that will contain the null-terminated character representation of the *datetime* component of interest. If *datetime* is NULL, *charbuf* will contain a zero-length string.

**datepart**

The date component of interest. Table 2-15 lists the date parts, the date part symbols recognized by DB-Library and the expected values. Note that the names of the months and the days in this table are those for English.

**Table 2-15: Date parts and their character representations  
(dbdatetime)**

| Date part   | Symbol    | Character representation of value |
|-------------|-----------|-----------------------------------|
| year        | DBDATE_YY | 1753 – 9999                       |
| quarter     | DBDATE_QQ | 1 – 4                             |
| month       | DBDATE_MM | January – December                |
| day of year | DBDATE_DY | 1 – 366                           |
| day         | DBDATE_DD | 1 – 31                            |
| week        | DBDATE_WK | 1 – 54 (for leap years)           |
| weekday     | DBDATE_DW | Monday – Sunday                   |
| hour        | DBDATE_HH | 0 – 23                            |
| minute      | DBDATE_MI | 0 – 59                            |
| second      | DBDATE_SS | 0 – 59                            |
| millisecond | DBDATE_MS | 0 – 999                           |

**datetime**

A pointer to the DBDATETIME value of interest.

**Return value**

The number of bytes placed into *\*charbuf*.

In case of error, dbdatetime returns -1.

**Usage**

- dbdatetime converts the specified component of a DBDATETIME structure into a character string.

- The names of the months and weekdays are in the language of the specified DBPROCESS. If *dbproc* is NULL, these names will be in DB-Library's default language.
- This function is very similar to the Transact-SQL *datetime* function.
- The following code fragment illustrates the use of *dbdatetime*:

```

dbcmd(dbproc, "select name, crdate from \
            master..sysdatabases");
dbsqlxexec(dbproc);
dbresults(dbproc);

while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
    /*
    ** Print the database name and its date info
    */
    dbconvert(dbproc, dbcoltype(dbproc, 2),
              dbdata(dbproc, 2), dbdatlen(dbproc, 2),
              SYBCHAR, datestring, -1);
    printf("%s: %s\n", (char *) (dbdata
                                (dbproc, 1)), datestring);

    /* Print the parts of the creation date */
    dbdatetime(dbproc, datestring, DBDATE_YY,
               (DBDATETIME *) (dbdata(dbproc, 2)));
    printf("\tYear = %s.\n", datestring);

    dbdatetime(dbproc, datestring, DBDATE_QQ,
               (DBDATETIME *) (dbdata(dbproc, 2)));
    printf("\tQuarter = %s.\n", datestring);

    dbdatetime(dbproc, datestring, DBDATE_MM,
               (DBDATETIME *) (dbdata(dbproc, 2)));
    printf("\tMonth = %s.\n", datestring);

    dbdatetime(dbproc, datestring, DBDATE_DW,
               (DBDATETIME *) (dbdata(dbproc, 2)));
    printf("\tDay of week = %s.\n", datestring);

    dbdatetime(dbproc, datestring, DBDATE_DD,
               (DBDATETIME *) (dbdata(dbproc, 2)));
    printf("\tDay of month = %s.\n", datestring);

    dbdatetime(dbproc, datestring, DBDATE_DY,
               (DBDATETIME *) (dbdata(dbproc, 2)));

```

```
printf("\tDay of year = %s.\n", datestring);

dbdatetime(dbproc, datestring, DBDATE_HH,
            (DBDATETIME *) (dbdata(dbproc, 2)));
printf("\tHour = %s.\n", datestring);

dbdatetime(dbproc, datestring, DBDATE_MI,
            (DBDATETIME *) (dbdata(dbproc, 2)));
printf("\tMinute = %s.\n", datestring);

dbdatetime(dbproc, datestring, DBDATE_SS,
            (DBDATETIME *) (dbdata(dbproc, 2)));
printf("\tSecond = %s.\n", datestring);

dbdatetime(dbproc, datestring, DBDATE_MS,
            (DBDATETIME *) (dbdata(dbproc, 2)));
printf("\tMillisecond = %s.\n", datestring);

dbdatetime(dbproc, datestring, DBDATE_WK,
            (DBDATETIME *) (dbdata(dbproc, 2)));
printf("\tWeek = %s.\n", datestring);
```

See also

dbconvert, dbdata, dbdatechar, dbdatecrack

## dbdateorder

|             |   |
|-------------|---|
| Description | Return the date component order for a given language.   |
| Syntax      | <pre>char *dbdateorder(dbproc, language)</pre><br><pre>DBPROCESS  *dbproc; char        *language;</pre>   |
| Parameters  | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>language</b></p> <p>The name of the language of interest.</p> |

|              |  |
|--------------|--|
| Return value | <p>A pointer to a null-terminated, 3-character string containing the characters “m,” “d,” and “y,” representing the month, day, and year date components, respectively. The order of the characters in the <code>dbdateorder</code> string corresponds to their order in <i>language</i>’s default datetime format.</p> <p><code>dbdateorder</code> returns a NULL pointer on failure.</p>   |
| Usage        | <ul style="list-style-type: none"> <li><code>dbdateorder</code> returns a character string that describes the order in which the month, day, and year date components appear in the specified language. If <i>language</i> is NULL, the current language of the specified DBPROCESS is used. If both <i>language</i> and <i>dbproc</i> are NULL, DB-Library’s default language is used.</li> </ul> <hr/> <p><b>Warning!</b> The date order string returned by <code>dbdateorder</code> is a pointer to DB-Library’s internal data structures. Application programs should neither modify this string, nor free it.</p> <hr/> <ul style="list-style-type: none"> <li>The following code fragment illustrates the use of <code>dbdateorder</code>:</li> </ul> <pre> /* Retrieve the date order from Adaptive Server Enterprise */ printf("date-order: %s\n",       (dbdateorder(DBPROCESS *)NULL, (char *)NULL)); </pre> |
| See also     | <code>dbconvert</code> , <code>dbdata</code> , <code>dbdatechar</code> , <code>dbdatecrack</code>  |

## dbdatepart

|             |   |
|-------------|---|
| Description | Return the specified part of a DBDATETIME value as a numeric value.   |
| Syntax      | <p>DBINT <code>dbdatepart</code>(<code>dbproc</code>, <code>datepart</code>, <code>datetime</code>)</p> <pre> DBPROCESS  *dbproc; int         datepart; DBDATETIME *datetime; </pre>  |
| Parameters  | <p><code>dbproc</code></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> |

datepart

The date component of interest. Table 2-16 lists the date parts, the date part symbols recognized by DB-Library and the expected values. Note that the names of the months and the days in this table are those for English.

**Table 2-16: Date parts and their character representations (dbdatepart)**

| Date part   | Symbol    | Character representation of value |
|-------------|-----------|-----------------------------------|
| year        | DBDATE_YY | 1753 – 9999                       |
| quarter     | DBDATE_QQ | 1 – 4                             |
| month       | DBDATE_MM | January – December                |
| day of year | DBDATE_DY | 1 – 366                           |
| day         | DBDATE_DD | 1 – 31                            |
| week        | DBDATE_WK | 1 – 54 (for leap years)           |
| weekday     | DBDATE_DW | Monday – Sunday                   |
| hour        | DBDATE_HH | 0 – 23                            |
| minute      | DBDATE_MI | 0 – 59                            |
| second      | DBDATE_SS | 0 – 59                            |
| millisecond | DBDATE_MS | 0 – 999                           |

datetime

A pointer to the DBDATETIME value of interest.

Return value

The value of the specified date part.

Usage

- dbdatepart returns the specified part of a DBDATETIME value as a numeric value.
- dbdatepart is similar to the Transact-SQL datepart function.

See also

dbconvert, dbdata, dbdatechar, dbdatecrack, dbdatetime

dbdatezero

Description

Initialize a DBDATETIME value to Jan 1, 1900 12:00:00:000AM.

Syntax

RETCODE dbdatezero(dbproc, dateptr)

DBPROCESS   \*dbproc;  
DBDATETIME   \*dateptr;



|              |   |
|--------------|---|
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL.</p> <p><b>dateptr</b><br/>A pointer to the DBDATETIME variable to initialize.</p> |
| Return value | <p>SUCCEED or FAIL.</p> <p>dbdatezero returns FAIL if <i>dateptr</i> is NULL.</p>   |
| Usage        | <ul style="list-style-type: none"><li>• dbdatezero initializes a DBDATETIME value to Jan 1, 1900 12:00:00:000AM.</li><li>• The range of legal DBDATETIME values is from January 1, 1753 to December 31, 9999. DBDATETIME values have a precision of 1/300th of a second (3.33 milliseconds).</li></ul>  |
| See also     | <p>dbdate4zero</p>  |

## dbdatlen

|              |   |
|--------------|---|
| Description  | <p>Return the length of the data in a regular result column.</p>  |
| Syntax       | <p>DBINT dbdatlen(dbproc, column)</p> <pre>DBPROCESS  *dbproc; int         column;</pre>  |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>column</b><br/>The number of the column of interest. The first column is number 1.</p> |
| Return value | <p>The length, in bytes, of the data that would be returned for the particular column. If the data has a null value, dbdatlen returns 0. If the column number is not in range, dbdatlen returns -1.</p>   |

## Usage

- This routine returns the length, in bytes, of data that would be returned by a select against a regular (that is, non-compute) result column. In most cases, this is the actual length of data for the column. For text and image columns, however, the integer returned by dbdatlen can be less than the actual length of data for the column. This is because the server global variable @@textsize limits the amount of text or image data returned by a select.
- Use the dbcollen routine to determine the maximum possible length for the data. Use dbdata to get a pointer to the data itself.
- Here is a small program fragment that uses dbdatlen:

```
DBPROCESS      *dbproc;
DBINT          row_number = 0;
DBINT          data_length;

/* Put the command into the command buffer */
dbcmd(dbproc, "select name from sysobjects");
/*
** Send the command to Adaptive Server Enterprise
and begin
** execution
*/
dbsqlxexec(dbproc);

/* Process the command results */
dbresults(dbproc);

/* Examine the data lengths of each row */
while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
    row_number++;
    data_length = dbdatlen(dbproc, 1);
    printf("row %ld, data length is %ld.\n",
           row_number, data_length);
}
```

## See also

dbcollen, dbcolname, dbcoltype, dbdata, dbnumcols

## dbdayname

## Description

Determine the name of a specified weekday in a specified language.

|              |  |
|--------------|--|
| Syntax       | <pre>char *dbdayname(dbproc, language, daynum)  DBPROCESS *dbproc; char *language; int daynum;</pre>   |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>language</b><br/>The name of the desired language.</p> <p><b>daynum</b><br/>The number of the desired day. Day numbers range from 1 (Monday) to 7 (Sunday).</p>   |
| Return value | The name of the specified day on success; a NULL pointer on error.   |
| Usage        | <ul style="list-style-type: none"> <li>• <code>dbdayname</code> returns the name of the specified day in the specified language. If <i>language</i> is NULL, <i>dbproc</i>'s current language is used. If both <i>language</i> and <i>dbproc</i> are NULL, then U.S. English is used.</li> <li>• The following code fragment illustrates the use of <code>dbdayname</code>: <pre>/* ** Retrieve the name of each day of the week in ** U.S. English. */ for (daynum = 1; daynum &lt;= 7; daynum++)     printf("Day %d: %s\n", daynum,         dbdayname((DBPROCESS *)NULL, (char *)NULL,             daynum));</pre> </li> </ul> |
| See also     | <code>db12hour</code> , <code>dbdateorder</code> , <code>dbmonthname</code> , <code>DBSETLNATLANG</code>   |

## DBDEAD

|             |  |
|-------------|--|
| Description | Determine whether a particular DBPROCESS is dead.    |
| Syntax      | <pre>DBBOOL DBDEAD(dbproc)  DBPROCESS *dbproc;</pre> |

|              |   |
|--------------|---|
| Parameters   | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>  |
| Return value | <p>“TRUE” or “FALSE.”</p>   |
| Usage        | <ul style="list-style-type: none"><li>• This macro indicates whether or not the specified DBPROCESS has been marked dead. It is particularly useful in user-supplied error handlers.</li><li>• If a DBPROCESS is dead, then almost every DB-Library routine that receives it as a parameter will immediately fail, calling the user-supplied error handler.</li></ul> <hr/> <p><b>Note</b> If there is no user-supplied error handler, a dead DBPROCESS will cause the affected DB-Library routines not to fail, but to abort.</p> <hr/> <ul style="list-style-type: none"><li>• Note that DBDEAD does not communicate with the server, but only checks the current status of a DBPROCESS. If a previously called DB-Library routine has not marked a DBPROCESS as dead, DBDEAD reports the DBPROCESS as healthy.</li></ul> |
| See also     | <p>dberrhandle, Errors on page 389</p>  |

## dberrhandle

|             |   |
|-------------|---|
| Description | <p>Install a user function to handle DB-Library errors.</p>   |
| Syntax      | <pre>int (*dberrhandle(handler))()  int      (*handler)();</pre>  |
| Parameters  | <p>handler</p> <p>A pointer to the user function that will be called whenever DB-Library determines that an error has occurred. DB-Library calls this function with six parameters shown in Table 2-17.</p> |

**Table 2-17: Error handler parameters**

| Parameter       | Meaning   |
|-----------------|---|
| <i>dbproc</i>   | The affected DBPROCESS. If there is no DBPROCESS associated with this error, this parameter will be NULL.   |
| <i>severity</i> | The severity of the error (datatype <i>int</i> ). Error severities are defined in <i>syberror.h</i> .   |
| <i>dberr</i>    | The identifying number of the error (datatype <i>int</i> ). Error numbers are defined in <i>sybdb.h</i> .   |
| <i>oserr</i>    | The operating-system-specific error number that describes the cause of the error (datatype <i>int</i> ). If there is no relevant operating system error, the value of this parameter will be DBNOERR. |
| <i>dberrstr</i> | A printable description of <i>dberr</i> (datatype <i>char *</i> ).  |
| <i>oserrstr</i> | A printable description of <i>oserr</i> (datatype <i>char *</i> ).  |

The error handler must return one of the four values listed in Table 2-18, directing DB-Library to perform particular actions:

**Table 2-18: Error handler returns**

| Return       | Action  |
|--------------|---|
| INT_EXIT     | Print an error message and abort the program. DB-Library will also return an error indication to the operating system. (Note to UNIX programmers: DB-Library will not leave a core file.)   |
| INT_CANCEL   | Return FAIL from the DB-Library routine that caused the error. Returning INT_CANCEL on timeout errors will kill the <i>dbproc</i> .   |
| INT_TIMEOUT  | Cancel the operation that caused the error but leave the <i>dbproc</i> in working condition. This return value is meaningful only for timeout errors (SYBETIME). In any other case, this value will be considered an error, and will be treated as an INT_EXIT.                 |
| INT_CONTINUE | Continue to wait for one additional timeout period. At the end of that period, call the error handler again. This return value is meaningful only for timeout errors (SYBETIME). In any other case, this value will be considered an error, and will be treated as an INT_EXIT. |

If the error handler returns any value besides these four, the program will abort.

Error handlers on the Windows platform must be declared with CS\_PUBLIC, as shown in the example below. For portability, callback handlers on other platforms should be declared CS\_PUBLIC as well.

The following example shows a typical error handler routine:

```
#include <sybfront.h>
```

```
#include <sybdb.h>
#include <syberror.h>

int CS_PUBLIC err_handler(dbproc, severity, dberr,
oserr, dberrstr, oserrstr)
DBPROCESS      *dbproc;
int             severity;
int             dberr;
int             oserr;
char            *dberrstr;
char            *oserrstr;
{
    if ((dbproc == NULL) || (DBDEAD(dbproc)))
        return(INT_EXIT);
    else
    {
        printf("DB-Library error:\n\t%s\n",
            dberrstr);
        if (oserr != DBNOERR)
            printf("Operating-system \
                error:\n\t%s\n", oserrstr);
        return(INT_CANCEL);
    }
}
```

**Return value**

A pointer to the previously installed error handler. This pointer is NULL if no error handler was installed before.

**Usage**

- dberrhandle installs an error-handler function that you supply. When a DB-Library error occurs, DB-Library will call this error handler immediately. You must install an error handler to handle DB-Library errors properly.
- If an application does not call dberrhandle to install an error-handler function, DB-Library ignores error messages. The messages are not printed.
- The user-supplied error handler will completely determine the response of DB-Library to any error that occurs. It must tell DB-Library whether to:
  - Abort the program, or
  - Return an error code and mark the DBPROCESS as “dead” (making it unusable), or
  - Cancel the operation that caused the error, or
  - Keep trying (in the case of a timeout error).

- If the user does not supply an error handler (or passes a NULL pointer to `dberrhandle`), DB-Library will exhibit its default error-handling behavior: It will abort the program if the error has made the affected DBPROCESS unusable (the user can call `DBDEAD` to determine whether or not a DBPROCESS has become unusable). If the error has not made the DBPROCESS unusable, DB-Library will simply return an error code to its caller.
- You can “de-install” an existing error handler by calling `dberrhandle` with a NULL parameter. You can also, at any time, install a new error handler. The new handler will automatically replace any existing handler.
- If the program refers to error severity values, its source file must include the header file called *syberror.h*.
- See Errors on page 389 for a list of DB-Library errors.
- Another routine, `dbmsghandle`, installs a message handler that DB-Library calls in response to the server error messages.
- If the application provokes messages from DB-Library and the server simultaneously, DB-Library calls the server message handler before it calls the DB-Library error handler.
- The DB-Library/C error value `SYBESMSG` is generated in response to a server error message, but not in response to a server informational message. This means that when a server error occurs, both the server message handler and the DB-Library/C error handler are called, but when the server generates an informational message, only the server message handler is called.

If you have installed a server message handler, you may want to write your DB-Library error handler so as to suppress the printing of any `SYBESMSG` error, to avoid notifying the user about the same error twice.

Table 2-19 provides information on when DB-Library/C calls an application’s message and error handlers:

**Table 2-19: Common errors**

| Error or message             | Message handler called? | Error handler called?  |
|------------------------------|-------------------------|--|
| SQL syntax error.            | Yes.                    | Yes ( <code>SYBESMSG</code> ).<br>(Code your handler to ignore the message.) |
| SQL print statement.         | Yes.                    | No.  |
| SQL <code>raiserror</code> . | Yes.                    | No.  |

| Error or message  | Message handler called?   | Error handler called?   |
|---|---|---|
| Server dies.  | No.   | Yes (SYBESEOF).<br>(Code your handler to exit the application.)                                     |
| Timeout from the server.  | No.   | Yes (SYBETIME).<br>(To wait for another timeout period, code your handler to return -INT_CONTINUE.) |
| <b>Note</b> The default timeout period is infinite. The error handler will not receive timeout notifications unless a timeout period is specified with dbsettime. |   |   |
| Deadlock on query.  | Yes.<br>(Code your handler to test for deadlock. See the dbsetuserdata on page 336 for an example.) | Yes (SYBESMSG).<br>(Code your handler to ignore the message.)                                       |
| Timeout on login.   | No.   | Yes (SYBEFCON, SYBECNN).  |
| Login fails (dbopen).   | Yes.  | Yes (SYBEPWD).<br>(Code your handler to exit the application.)                                      |
| Use database message.   | Yes.<br>(Code your handler to ignore the message.)  | No.   |
| Incorrect use of DB-Library/C calls, such as not calling dbresults when required.   | No.   | Yes (SYBERPND, ...) Yes (SYBERPND, .)   |
| Fatal Server error (severity greater than 16).  | Yes.<br>(Code your handler to exit the application.)  | Yes (SYBESMSG).   |

See also DBDEAD, dbmsghandle, Errors on page 389

## dbexit

|              |   |
|--------------|---|
| Description  | Close and deallocate all DBPROCESS structures, and clean up any structures initialized by dbinit. |
| Syntax       | void dbexit()   |
| Return value | None.   |



|          |   |
|----------|---|
| Usage    | <ul style="list-style-type: none"> <li>• <code>dbexit</code> calls <code>dbcclose</code> repeatedly for all allocated <code>DBPROCESS</code> structures. <code>dbcclose</code> cleans up any activity associated with a single <code>DBPROCESS</code> structure and deallocates the space.</li> <li>• You can use <code>dbcclose</code> directly to close just a single <code>DBPROCESS</code> structure.</li> <li>• <code>dbexit</code> also cleans up any structures initialized by <code>dbinit</code>, releasing the memory associated with those structures. It must be the last DB-Library call in any application that calls <code>dbinit</code>.</li> <li>• To ensure future compatibility and portability, Sybase strongly recommends that all applications call <code>dbinit</code> and <code>dbexit</code>, no matter what their environment.</li> </ul> <p>For environments requiring <code>dbinit</code>, the application must not make any other DB-Library call after calling <code>dbexit</code>.</p> |
| See also | <code>dbcclose</code> , <code>dbinit</code> , <code>dbopen</code>   |

## dbfcmd

|             |  |
|-------------|--|
| Description | Add text to the <code>DBPROCESS</code> command buffer using C runtime library <code>sprintf</code> -type formatting.   |
| Syntax      | <pre>RETCODE dbfcmd(dbproc, cmdstring, args...)</pre> <pre>DBPROCESS  *dbproc; char        *cmdstring; ???         args...;</pre>  |
| Parameters  | <p><b>dbproc</b><br/>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>cmdstring</b><br/>A format string of the form used by the <code>sprintf</code> routine.</p> <p>There is an optional and variable number of arguments to <code>dbfcmd</code>. The number and type of arguments required depends on the format specifiers included in the <i>cmdstring</i> argument. The arguments are passed directly to the C-library <code>sprintf</code> function. Neither <code>dbfcmd</code> nor the C compiler can type check these arguments. As with using <code>sprintf</code>, the programmer must ensure that each argument type matches the corresponding format specifier.</p> |

Return value

SUCCEED or FAIL.

Usage

- This routine adds text to the Transact-SQL command buffer in the DBPROCESS structure. dbfcmd works just like the sprintf function in the C language standard I/O library, using % conversion specifiers. If you do not need any of the formatting capability of sprintf, you can use dbcmd instead.
- Table 2-20 lists the conversions supported by dbfcmd:

**Table 2-20: dbfcmd conversions**

| Conversion | Program variable type                                      |
|------------|--|
| %s         | char*, null-terminated                                     |
| %d         | int, decimal representation                                |
| %f         | double   |
| %g         | double   |
| %e         | double   |
| %%         | None, the “%” character is written into the command buffer |

The datatype SYBDATETIME must be converted to a character string and passed using %s. The datatype SYBMONEY may be converted to a character string and passed using %s, or converted to float and passed using %f.

---

**Note** Currently, only eight arguments may be handled in each call to dbfcmd. To format commands that require more than eight arguments, call dbfcmd repeatedly.

---

- dbfcmd manages the space allocation for the command buffer. It adds to the existing command buffer—it does not delete or overwrite the current contents except after the buffer has been sent to the server (see “Clearing the command buffer” on page 151). A single command buffer may contain multiple commands; in fact, this represents an efficient use of the command buffer.
- The application may call dbfcmd repeatedly. The command strings in sequential calls are just concatenated together. It is the program’s responsibility to ensure that any necessary blanks appear between the end of one string and the beginning of the next.
- Here is a small program fragment that uses dbfcmd to build up a multiline SQL command:

```
char          *column_name;  
DBPROCESS    *dbproc;
```

```

int          low_id;
char         *object_type;
char         *tablename;
dbfcmd(dbproc, "select %s from %s", column_name,
        tablename);
dbfcmd(dbproc, " where id > %d", low_id);
dbfcmd(dbproc, " and type='%s'", object_type);

```

Note the required spaces at the start of the second and third command strings.

- When passing character or string variables to `dbfcmd`, beware of variables that contain quotes (single or double) or null characters (ASCII 0).
  - Improperly placed quotes in the SQL command can cause SQL syntax errors or, worse yet, unanticipated query results.
  - NULL characters (ASCII 0) should never be inserted into the command buffer. They can confuse DB-Library and the server, causing SQL syntax errors or unanticipated query results.
- Since `dbfcmd` calls `sprintf`, you must remember that `%` (percentage sign) has a special meaning as the beginning of a format command. If you want to include `%` in the command string, you must precede it with another `%`.
- Be sure to guard against passing a null pointer as a string parameter to `dbfcmd`. If a null value is a possibility, you should check for it before using the variable in a `dbfcmd` call.
- The application can intermingle calls to `dbcmd` and `dbfcmd`.
- At any time, the application can access the contents of the command buffer through calls to `dbgetchar`, `dbstrlen`, and `dbstrcpy`.
- Available memory is the only constraint on the size of the DBPROCESS command buffer created by calls to `dbcmd` and `dbfcmd`.

#### Clearing the command buffer

After a call to `dbsqlexec` or `dbsqlsend`, the first call to either `dbcmd` or `dbfcmd` automatically clears the command buffer before the new text is entered. If this situation is undesirable, set the `DBNOAUTOFREE` option. When `DBNOAUTOFREE` is set, the command buffer is cleared only by an explicit call to `dbfreebuf`.

### Limitations

Currently, only eight *args* may be handled in each call to `dbcmd`. To format commands that require more than eight *args*, call `dbcmd` repeatedly. On some platforms, `dbcmd` may allow more than eight *args* per call. For portable code, do not pass more than eight arguments.

Because it makes text substitutions, `dbcmd` uses a working buffer in addition to the `DBPROCESS` command buffer. `dbcmd` allocates this working buffer dynamically. The size of the space it allocates is equal to the maximum of a defined constant (1024) or the string length of *cmdstring* \*2 . For example, if the length of *cmdstring* is 600 bytes, `dbcmd` allocates a working buffer 1200 bytes long. If the length of *cmdstring* is 34 bytes, `dbcmd` allocates a working buffer 1024 bytes long. To work around this limitation:

```
sprintf (buffer, "%s", SQL command");  
dbcmd (dbproc, buffer)
```

If the *args* are very big in comparison to the size of *cmdstring*, the working buffer may not be large enough to hold the string after substitutions are made. In this situation, break *cmdstring* up and use multiple calls to `dbcmd`.

Note that the working buffer is not the same as the `DBPROCESS` command buffer. The working buffer is a temporary buffer used only by `dbcmd` when making text substitutions. The `DBPROCESS` command buffer holds the text after substitutions have been made. There is no constraint, other than available memory, on the size of the `DBPROCESS` command buffer.

See also

`dbcmd`, `dbfreebuf`, `dbgetchar`, `dbstrcpy`, `dbstrlen`, Options on page 407

## DBFIRSTROW

Description

Return the number of the first row in the row buffer.

Syntax

```
DBINT DBFIRSTROW(dbproc)
```

`DBPROCESS`    *\*dbproc*;

Parameters

*dbproc*

A pointer to the `DBPROCESS` structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

|              |  |
|--------------|--|
| Return value | The number of the first row in the row buffer. Rows are counted from the first row returned from the server, whose number is 1. This routine returns 0 if there is an error.   |
| Usage        | <ul style="list-style-type: none"><li>• This macro returns the number of the first row in the row buffer.</li><li>• If you are not buffering rows, DBFIRSTROW, DBCURROW, and DBLASTROW always have the same value. If you have allowed buffering by setting the DBBUFFER option, DBFIRSTROW returns the number of the first row in the row buffer.</li><li>• Note that the first row returned from the server (whose value is 1) is not necessarily the first row in the row buffer. The rows in the row buffer are dependent on manipulation by the application program. See the dbclrbuf reference page for details.</li></ul> |
| See also     | dbclrbuf, DBCURROW, dbgetrow, DBLASTROW, dbnextrow, dbsetopt, Options on page 407  |

## dbfree\_xlate

|             |   |
|-------------|---|
| Description | Free a pair of character set translation tables.  |
| Syntax      | <pre>RETCODE *dbfree_xlate(dbproc, xlt_tosrv, xlt_todisp)</pre><br><pre>DBPROCESS  *dbproc;<br/>DBXLATE    *xlt_tosrv;<br/>DBXLATE    *xlt_todisp;</pre>  |
| Parameters  | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>xlt_tosrv</b><br/>A pointer to a translation table used to translate display-specific character strings to the server character strings. The translation table is allocated using dbload_xlate.</p> <p><b>xlt_todisp</b><br/>A pointer to a translation table used to translate server character strings to display-specific character strings. The translation table is allocated using dbload_xlate.</p> |

|              |  |
|--------------|--|
| Return value | SUCCEED or FAIL.   |
| Usage        | <ul style="list-style-type: none"><li>• This routine frees a pair of character set translation tables allocated by <code>dbload_xlate</code>.</li><li>• Character set translation tables translate characters between the server's standard character set and the display device's character set.</li><li>• The following code fragment illustrates the use of <code>dbfree_xlate</code><pre>char      destbuf[128]; int       srcbytes_used; DBXLATE   *xlt_todisp;  DBXLATE   *xlt_tosrv; dbload_xlate((DBPROCESS *)NULL, "iso_1", "trans.xlt", &amp;xlt_tosrv, &amp;xlt_todisp); printf("Original string: \n\t%s\n\n",       TEST_STRING); dbxlate((DBPROCESS *)NULL, TEST_STRING,       strlen(TEST_STRING), destbuf, -1, xlt_todisp,       &amp;srcbytes_used); printf("Translated to display character set: \       \n\t%s\n\n", destbuf); dbfree_xlate((DBPROCESS *)NULL, xlt_tosrv,       xlt_todisp);</pre></li></ul> |
| See also     | <code>dbload_xlate</code> , <code>dbxlate</code>   |

## dbfreebuf

|              |  |
|--------------|--|
| Description  | Clear the command buffer.  |
| Syntax       | <pre>void dbfreebuf(dbproc)</pre><br><pre>DBPROCESS  *dbproc;</pre>  |
| Parameters   | <p><code>dbproc</code></p> <p>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>                           |
| Return value | None.  |
| Usage        | <ul style="list-style-type: none"><li>• This routine clears a <code>DBPROCESS</code> command buffer by freeing any space allocated to it. It then sets the command buffer to <code>NULL</code>. Commands are added to the command buffer with the <code>dbcmd</code> or <code>dbfcmd</code> routine.</li></ul> |

- After a call to `dbsqlxexec` or `dbsqlsend`, the first call to either `dbcmd` or `dbfcmd` automatically calls `dbfreebuf` to clear the command buffer before the new text is entered. If this situation is undesirable, set the `DBNOAUTOFREE` option. When `DBNOAUTOFREE` is set, the command buffer is cleared only by an explicit call to `dbfreebuf`.
- At any time, the application can access the contents of the command buffer through calls to `dbgetchar`, `dbstrlen`, and `dbstrcpy`.

See also

`dbcmd`, `dbfcmd`, `dbgetchar`, `dbsqlxexec`, `dbsqlsend`, `dbstrcpy`, `dbstrlen`, Options on page 407

## dbfreequal

Description Free the memory allocated by `dbqual`.

Syntax `void dbfreequal(qualptr)`

`char *qualptr;`

Parameters `qualptr`

A pointer to the memory allocated by `dbqual`.

Return value None.

Usage

- `dbfreequal` is one of the DB-Library browse mode routines. See Chapter 1, “Introducing DB-Library” for a detailed discussion of browse mode.
- `dbqual` provides a `where` clause that the application can use to update a single row in a browsable table. In doing so, it dynamically allocates a buffer to contain the `where` clause. When the `where` clause is no longer needed, the application can use `dbfreequal` to deallocate the buffer.

See also

`dbqual`

## dbfreesort

Description Free a sort order structure allocated by `dbloadsort`.

Syntax `RETCODE dbfreesort(dbproc, sortorder)`

`DBPROCESS *dbproc;`  
`DBSORTORDER *sortorder;`

|              |   |
|--------------|---|
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>sortorder</b></p> <p>A pointer to a DBSORTORDER structure allocated through dbloadsort.</p>   |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"><li>• dbfreesort frees a sort order structure that was allocated using dbloadsort. DB-Library routines such as dbstrcmp and dbstrsort use sort orders to determine how character data must be sorted.</li><li>• When an application program does sorting or comparing, it automatically sorts character data the same way the server does. If no sort order has been loaded, routines such as dbstrcmp and dbstrsort sort characters by their binary values.</li></ul> <hr/> <p><b>Warning!</b> Application programs must not attempt to use operating-system facilities to free the <i>*sortorder</i> structure directly, as it may have been allocated using some mechanism other than malloc (on operating systems where malloc is not supported), and it may consist of multiple parts, some of which must be freed separately.</p> <hr/> <ul style="list-style-type: none"><li>• The following code fragment illustrates the use of dbfreesort:</li></ul> <pre>sortorder = dbloadsort(dbproc);  retval = dbstrcmp(dbproc, "ABC", 3, "abc", 3,                  sortorder); printf("ABC dbstrcmp'ed with abc yields %d.\n",        retval);  retval = dbstrcmp(dbproc, "abc", 3, "ABC", 3,                  sortorder); printf("abc dbstrcmp'ed with ABC yields %d.\n",        retval);  dbfreesort(dbproc, sortorder);</pre> |
| See also     | dbloadsort, dbstrcmp, dbstrsort   |



## dbgetchar

|              |  |
|--------------|--|
| Description  | Return a pointer to a character in the command buffer.   |
| Syntax       | <pre>char *dbgetchar(dbproc, n)</pre><br><pre>DBPROCESS  *dbproc;<br/>int        n;</pre>  |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>n</b><br/>The position of the desired character in the command buffer. The first character position is 0.</p>   |
| Return value | dbgetchar returns a pointer to the <i>n</i> th character in the command buffer. If <i>n</i> is not in range, dbgetchar returns NULL.   |
| Usage        | <ul style="list-style-type: none"><li>• You can use dbgetchar to retrieve a pointer to a particular character in the command buffer. dbgetchar returns a pointer to a character in the command buffer whose position is indicated by <i>n</i>. The first character has position 0.</li><li>• Internally, the command buffer is a linked list of non-null-terminated text strings. dbgetchar, dbstrcpy, and dbstrlen together provide a way to locate and copy parts of the command buffer.</li><li>• Since the command buffer is not just one large text string, but rather a linked list of text strings, you must use dbgetchar to index through the buffer. If you just get a pointer using dbgetchar and then increment it yourself, it will probably fall off the end of a string and cause a segmentation fault.</li></ul> |
| See also     | dbcmd, dbfcmd, dbfreebuf, dbstrcpy, dbstrlen   |

## dbgetcharset

|             |   |
|-------------|---|
| Description | Get the name of the client character set from the DBPROCESS structure.  |
| Syntax      | <pre>char *dbgetcharset(dbproc)</pre><br><pre>DBPROCESS  *dbproc;</pre> |

|              |   |
|--------------|---|
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and server.</p>   |
| Return value | <p>A pointer to the null-terminated name of the client character set, or NULL in case of error.</p>   |
| Usage        | <ul style="list-style-type: none"><li>• <b>dbgetcharset</b> returns the name of the client's character set.</li><li>• DB-Library/C clients can use a different character set than the server or servers to which they are connected. If a client and server are using different character sets, and the server supports character translation for the client's character set, it will perform all conversions to and from its own character set when communicating with the client.</li><li>• An application can inform the server what character set it is using through <b>DBSETLCHARSET</b>.</li><li>• To determine if the server is performing character set translations, an application can call <b>dbcharsetconv</b>.</li><li>• To get the name of the server character set, an application can call <b>dbservcharset</b>.</li></ul> |
| See also     | <p><b>dbcharsetconv</b>, <b>dblogin</b>, <b>dbopen</b>, <b>dbservcharset</b>, <b>DBSETLCHARSET</b></p>  |

## dbgetloginfo

|             |   |
|-------------|---|
| Description | <p>Transfer Tabular Data Stream (TDS) login response information from a DBPROCESS structure to a newly allocated DBLOGININFO structure.</p>   |
| Syntax      | <p><b>RETCODE dbgetloginfo(dbproc, loginfo)</b></p> <pre>DBPROCESS    *dbproc;<br/>DBLOGININFO  **loginfo;</pre>  |
| Parameters  | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and the server.</p> |

**loginfo**

The address of a DBLOGINFO pointer variable. dbgetloginfo sets the DBLOGINFO pointer to the address of a newly-allocated DBLOGINFO structure.

**Return value**

SUCCEED or FAIL.

**Usage**

- dbgetloginfo transfers TDS login response information from a DBPROCESS structure to a newly allocated DBLOGINFO structure.
- An application needs to call dbgetloginfo only if 1) it is an Open Server gateway application, and 2) it is using TDS passthrough.
- TDS is an application protocol used for the transfer of requests and request results between clients and servers.
- When a client connects directly to a server, the two programs negotiate the TDS format they will use to send and receive data. When a gateway application uses TDS passthrough, the application forwards TDS packets between the client and a remote server without examining or processing them. For this reason, the remote server and the client must agree on a TDS format to use.
- dbgetloginfo is the second of four calls, two of them Server Library calls, that allow a client and remote server to negotiate a TDS format. The calls, which can be made only in a SRV\_CONNECT event handler, are:
  - srv\_getloginfo - allocate a DBLOGINFO structure and fill it with TDS information from a client SRV\_PROC.
  - dbsetloginfo - transfer the TDS information retrieved in step 1 from the DBLOGINFO structure to a DB-Library/C LOGINREC structure, and then free the DBLOGINFO structure. After the information is transferred, the application can use this LOGINREC structure in the dbopen call which establishes its connection with the remote server.
  - dbgetloginfo - transfer the remote server's response to the client's TDS information from a DBPROCESS structure into a newly-allocated DBLOGINFO structure.
  - srv\_setloginfo - send the remote server's response, retrieved in the previous step, to the client, and then free the DBLOGINFO structure.
- This is an example of a SRV\_CONNECT handler preparing a remote connection for TDS passthrough:

```
RETCODE connect_handler(srvproc)
SRVPROC      *srvproc;
{
```

```
DBLOGININFO    *loginfo;
LOGINREC       *loginrec;
DBPROCESS      *dbproc;
/*
** Get the TDS login information from the client
** SRV_PROC.
**/
srv_getloginfo(srvproc, &loginfo);
/* Get a LOGINREC structure */
loginrec = dblogin();
/*
** Initialize the LOGINREC with the login info
** from the SRV_PROC.
**/
dbsetloginfo(loginrec, loginfo);
/* Connect to the remote server */
dbproc = dbopen(loginrec, REMOTE_SERVER_NAME);
/*
** Get the TDS login response information from
** the remote connection.
**/
dbgetloginfo(dbproc, &loginfo);
/*
** Return the login response information to the
** SRV_PROC.
**/
srv_setloginfo(srvproc, loginfo);
/* Accept the connection and return */
srv_senddone(srvproc, 0, 0, 0);
return(SRV_CONTINUE);
}
```

See also

dbrecvpassthru, dbsendpassthru, dbsetloginfo

## dbgetlusername

Description                      Return the user name from a LOGINREC structure.

Syntax                            int dbgetlusername(login, name\_buffer, buffer\_len)

```
LOGINREC    *login;
BYTE        *name_buffer;
int         buffer_len;
```

|              |  |
|--------------|--|
| Parameters   | <code>login</code><br>A pointer to a LOGINREC structure, which can be passed as an argument to <code>dbopen</code> . You can get a LOGINREC structure by calling <code>dblogin</code> .  |
|              | <code>name_buffer</code><br>A pointer to a buffer. The user name will be copied from the LOGINREC structure to this buffer.  |
|              | <code>buffer_len</code><br>The length, in bytes, of the destination buffer.  |
| Return value | The number of bytes copied into the destination buffer, not including the null-terminator.   |
|              | If the user name is more than <i>buffer_len</i> - 1 bytes long, <code>dbgetusername</code> copies <i>buffer_len</i> - 1 bytes into the destination buffer and returns DBTRUNCATED.<br><br><code>dbgetusername</code> returns FAIL if <i>login</i> is NULL, <i>name_buffer</i> is NULL, or <i>buffer_len</i> is less than 0.  |
| Usage        | <ul style="list-style-type: none"><li>• <code>dbgetusername</code> copies the user name from LOGINREC structure into the <i>name_buffer</i> buffer.</li><li>• To set the user name in a LOGINREC structure, use DBSETLUSER.</li><li>• <code>dbgetusername</code> copies a maximum of <i>buffer_len</i> - 1 bytes, and null-terminates the user name string. Since the longest user name in a LOGINREC structure is DBMAXNAME bytes, an application will never need a destination buffer longer than DBMAXNAME + 1 bytes.</li><li>• If the user name in the LOGINREC is longer than <i>buffer_len</i> - 1 bytes, <code>dbgetusername</code> truncates the name and returns DBTRUNCATED.</li></ul> |
| See also     | <code>dblogin</code> , DBSETLUSER  |

## dbgetmaxprocs

|              |   |
|--------------|---|
| Description  | Determine the current maximum number of simultaneously open DBPROCESSes.                    |
| Syntax       | <code>int dbgetmaxprocs()</code>  |
| Parameters   | None.   |
| Return value | An integer representing the current limit on the number of simultaneously open DBPROCESSes. |

|          |  |
|----------|--|
| Usage    | A DB-Library program has a maximum number of simultaneously open DBPROCESSes. By default, this number is 25. The application program may change this limit by calling <code>dbsetmaxprocs</code> . |
| See also | <code>dbopen</code> , <code>dbsetmaxprocs</code>   |

## dbgetnatlang

|              |  |
|--------------|--|
| Description  | Get the national language from the DBPROCESS structure.  |
| Syntax       | <pre>char* dbgetnatlang(dbproc)  DBPROCESS  *dbproc;</pre>   |
| Parameters   | <p><code>dbproc</code></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and server.</p>  |
| Return value | A pointer to a character string representing the national language that the client DBPROCESS is using.   |
| Usage        | <ul style="list-style-type: none"><li>• <code>dbgetnatlang</code> returns a pointer to the name of the national language that a client is using.</li><li>• DB-Library/C clients may use a different national language than the server or servers to which they are connected. An application can inform the server what national language it wishes to use through <code>DBSETLNATLANG</code>.</li></ul> |
| See also     | <code>dblogin</code> , <code>dbopen</code> , <code>DBSETLNATLANG</code>  |

## dbgetoff

|             |  |
|-------------|--|
| Description | Check for the existence of Transact-SQL constructs in the command buffer.  |
| Syntax      | <pre>int dbgetoff(dbproc, offtype, startfrom)  DBPROCESS  *dbproc; DBUSMALLINT offtype; int         startfrom;</pre> |

|              |   |
|--------------|---|
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>offtype</b><br/>The type of offset you want to find. The types, which are defined in the header file <i>sybdb.h</i>, are:</p> <p>OFF_SELECT<br/>OFF_FROM<br/>OFF_ORDER<br/>OFF_COMPUTE<br/>OFF_TABLE<br/>OFF_PROCEDURE<br/>OFF_STATEMENT<br/>OFF_PARAM<br/>OFF_EXEC</p> <p>See Options on page 407 for details.</p> <p><b>startfrom</b><br/>The point in the buffer at which to start looking. The command buffer begins at 0.</p> |
| Return value | The character offset into the command buffer for the specified offset. If the offset is not found, -1 is returned.  |
| Usage        | <ul style="list-style-type: none"> <li>If the DBOFFSET option has been set (see Options on page 407), this routine can check for the location of certain Transact-SQL constructs in the command buffer. As a simple example, assume the program does not know the contents of the command buffer but needs to know where the SQL keyword <code>select</code> appears:</li> </ul> <pre> int      select_offset[10]; int      last_offset; int      i; /* Set the offset option */ dbsetopt(dbproc, DBOFFSET, "select");  /* ** Assume the command buffer contains the ** following selects. **/ dbcmd(dbproc, "select x = 100 select y = 5");  /* Send the query to Adaptive Server Enterprise */ </pre> |

```
        dbsqllexec(dbproc);

/* Get all the offsets to the select keyword */
for (i = 0, last_offset = 0; last_offset != -1;
    i++)
    if ((last_offset = dbgetoff(dbproc,
        OFF_SELECT, last_offset) != -1)
        select_offset[i] = last_offset++;
```

In this example, select\_offset[0] = 0 and select\_offset[1] = 15.

- dbgetoff does not recognize select statements in a subquery. Thus, if the command buffer contained:

```
select pub_name
from publishers
where pub_id not in
    (select pub_id
     from titles
     where type = "business")
```

the second “select” would not be recognized.

See also

dbcmd, dbgetchar, dbsetopt, dbstrcpy, dbstrlen, Options on page 407

## dbgetpacket

|              |  |
|--------------|--|
| Description  | Return the TDS packet size currently in use.   |
| Syntax       | <pre>int dbgetpacket(dbproc)</pre><br><pre>DBPROCESS *dbproc;</pre>  |
| Parameters   | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and the server.</p> |
| Return value | The TDS packet size currently in use.  |
| Usage        | <ul style="list-style-type: none"><li>• dbgetpacket returns the TDS packet size currently in use.</li><li>• TDS (Tabular Data Stream) is an application protocol used for the transfer of requests and request results between clients and servers.</li></ul>    |



- TDS data is sent in fixed-size chunks, called “packets”. TDS packets have a default size of 512 bytes.
- An application may change the TDS packet size using DBSETLPACKET, which sets the packet size field in the LOGINREC structure. When the application logs in to the server or Open Server, the server sets the TDS packet size for the created DBPROCESS connection to be equal to or less than the value of this field. The packet size is set to a value less than the value of the field if the server is experiencing space constraints. Otherwise, the packet size will be equal to the value of the field.
- If an application sends or receives large amounts of text or image data, a packet size larger than the default 512 bytes may improve efficiency, since it results in fewer network reads and writes.

See also

DBSETLPACKET

## dbgetrow

Description

Read the specified row in the row buffer.

Syntax

STATUS dbgetrow(dbproc, row)

```
DBPROCESS  *dbproc;
DBINT      row;
```

Parameters

**dbproc**

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

**row**

The number of the row to read. Rows are counted from the first row returned from the server, whose number is 1. Note that the first row in the row buffer is not necessarily the first row returned from the server.

Return value

dbgetrow can return four different types of values:

- If the current row is a regular row, REG\_ROW is returned.
- If the current row is a compute row, the *computeid* of the row is returned. (See the dbaltbind reference page for information on the *computeid*.)

## Usage

- If the row is not in the row buffer, `NO_MORE_ROWS` is returned, and the current row is left unchanged.
- If the routine was unsuccessful, `FAIL` is returned.
- `dbgetrow` sets the current row in the row buffer to a specific row and reads it. This routine works only if the `DBBUFFER` option is on, enabling row buffering. When `dbgetrow` is called, any binding of row data to program variables (as specified with `dbbind` or `dbaltbind`) takes effect.
- Row buffering provides a way to keep a specified number of server result rows in program memory. Without row buffering, the result row generated by each new `dbnextrow` call overwrites the contents of the previous result row. Row buffering is therefore useful for programs that need to look at result rows in a non-sequential manner. It does, however, carry a memory and performance penalty because each row in the buffer must be allocated and freed individually. Therefore, use it only if you need to. Specifically, the application should only turn the `DBBUFFER` option on if it calls `dbgetrow` or `dbsetrow`. Note that row buffering has nothing to do with network buffering and is a completely independent issue.
- When row buffering is not allowed, the application processes each row as it is read from the server, by calling `dbnextrow` repeatedly until it returns `NO_MORE_ROWS`. When row buffering is enabled, the application can use `dbgetrow` to jump to any row that has already been read from the server with `dbnextrow`. Subsequent calls to `dbnextrow` cause the application to read successive rows in the buffer. When `dbnextrow` reaches the last row in the buffer, it reads rows from the server again, if there are any. Once the buffer is full, `dbnextrow` does not read any more rows from the server until some of the rows have been cleared from the buffer with `dbclrbuf`.
- The macros `DBFIRSTROW`, `DBLASTROW`, and `DBCURROW` are useful in conjunction with `dbgetrow` calls. `DBFIRSTROW`, for instance, gets the number of the first row in the buffer. Thus, the call:  

```
dbgetrow(dbproc, DBFIRSTROW(dbproc))
```

sets the current row to the first row in the buffer.
- The routine `dbsetrow` sets a buffered row to “current” but does not read the row.
- For an example of row buffering, see the sample program *example4.c*.

## See also

`dbaltbind`, `dbbind`, `dbclrbuf`, `DBCURROW`, `DBFIRSTROW`, `DBLASTROW`, `dbnextrow`, `dbsetrow`, Options on page 407

## DBGETTIME

|              |   |
|--------------|---|
| Description  | Return the number of seconds that DB-Library will wait for a server response to a SQL command.  |
| Syntax       | int DBGETTIME()   |
| Return value | The timeout value—the number of seconds that DB-Library waits for a server response before timing out. A timeout value of 0 represents an infinite timeout period.  |
| Usage        | <ul style="list-style-type: none"><li>• This routine returns the length of time in seconds that DB-Library will wait for a server response during calls to <code>dbsqlexec</code>, <code>dbsqllok</code>, <code>dbresults</code>, and <code>dbnextrow</code>. The default timeout value is 0, which represents an infinite timeout period.</li><li>• The program can call <code>dbsettime</code> to change the timeout value.</li></ul> |
| See also     | <code>dbsettime</code>  |

## dbgetuserdata

|              |  |
|--------------|--|
| Description  | Return a pointer to user-allocated data from a <code>DBPROCESS</code> structure.   |
| Syntax       | <pre>BYTE *dbgetuserdata(dbproc)  DBPROCESS  *dbproc;</pre>  |
| Parameters   | <p><code>dbproc</code></p> <p>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> |
| Return value | A generic <code>BYTE</code> pointer to the user's private data space. This pointer must have been previously saved with the <code>dbsetuserdata</code> routine.  |
| Usage        | <ul style="list-style-type: none"><li>• This routine returns, from a <code>DBPROCESS</code> structure, a pointer to user-allocated data. The application must have previously saved this pointer with the <code>dbsetuserdata</code> routine.</li></ul>                              |

- dbgetuserdata and dbsetuserdata allow the application to associate user data with a particular DBPROCESS. This avoids the necessity of using global variables for this purpose. One use for these routines is to handle deadlock, as shown in the example on the dbsetuserdata reference page. That example reruns the transaction when the application's message handler detects deadlock.
- This routine is particularly useful when the application has multiple DBPROCESSes.

See also

dbsetuserdata

## dbhasretstat

|              |   |
|--------------|---|
| Description  | Determine whether the current command or remote procedure call generated a return status number.  |
| Syntax       | <pre>DBBOOL dbhasretstat(dbproc)</pre><br><pre>DBPROCESS  *dbproc;</pre>  |
| Parameters   | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>  |
| Return value | “TRUE” or “FALSE”.  |
| Usage        | <ul style="list-style-type: none"><li>• This routine determines whether the current Transact-SQL command or remote procedure call generated a return status number. Status numbers are returned by all stored procedures running on Adaptive Server Enterprise. Since status numbers are a feature of stored procedures, only a remote procedure call or an execute command can generate a status number.</li><li>• The dbretstatus routine actually gets the status number. Stored procedures that complete normally return a status number of 0. For a list of return status numbers, see the <i>Adaptive Server Enterprise Reference Manual</i>.</li></ul> |

- When executing a stored procedure, the server returns the status number immediately after returning all other results. Therefore, the application can call `dbhasretstat` only after processing the stored procedure's results by calling `dbresults`, as well as `dbnextrow` if appropriate. (Note that a stored procedure can generate several sets of results—one for each `select` it contains.) Before the application can call `dbhasretstat` or `dbretstatus`, it must call `dbresults` and `dbnextrow` as many times as necessary to process all the results.
- The order in which the application processes the status number and any return parameter values is unimportant.
- When a stored procedure has been executed as an RPC command using `dbrpcinit`, `dbrpcparam`, and `dbrpcsend`, then the return status can be retrieved after all other results have been processed. For an example of this usage, see the sample program *example8.c*.
- When a stored procedure has been executed from a batch of Transact-SQL commands (with `dbsqlxexec` or `dbsqlsend`), then other commands might execute after the stored procedure. This situation makes return-status retrieval a little more complicated.
  - If you are sure that the stored procedure command is the only command in the batch, then you can retrieve the return status after the `dbresults` loop, as shown in the sample program *example8.c*.
  - If the batch can contain multiple commands, then the return status should be retrieved inside the `dbresults` loop, after all rows have been fetched with `dbnextrow`. The code below shows the program logic to retrieve the return status value in this situation.

```

while ( (result_code = dbresults(dbproc)
        != NO_MORE_RESULTS)
{
    if (result_code == SUCCEED)
    {
        ... bind rows here ...
        while ((row_code = dbnextrow(dbproc))
            != NO_MORE_ROWS)
        {
            ... process rows here ...
        }
        /* Now check for a return status */
        if (dbhasretstat(dbproc) == TRUE)
        {
            printf("(return status %d)\n",
                dbretstatus(dbproc));

```

```
    }
    if (dbnumrets(dbproc) > 0)
    {
        ... get output parameters here ...
    }
} /* if result_code */
else
{
    printf("Query failed.\n");
}
} /* while dbresults */
```

See also dbnextrow, dbresults, dbretdata, dbretstatus, dbrpcinit, dbrpcparam, dbrpcsend

## dbinit

Description

Initialize DB-Library.

Syntax

RETCODE dbinit()

Return value

SUCCEED or FAIL.

Usage

- This routine initializes certain private DB-Library structures. For environments that require it, the application must call dbinit before calling any other DB-Library routine. Most DB-Library routines will cause the application to exit if they are called before dbinit.
- To ensure future compatibility and portability, Sybase strongly recommends that all applications call dbinit, no matter what their operating environment.

See also

dbexit

## DBIORDESC

Description

(UNIX only) Provide program access to the UNIX file descriptor used by a DBPROCESS to read data coming from the server.

Syntax

int DBIORDESC(dbproc)

DBPROCESS \*dbproc;

|              |  |
|--------------|--|
| Parameters   | <p><code>dbproc</code></p> <p>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>   |
| Return value | <p>An integer file descriptor used by the specified <code>DBPROCESS</code> to read data coming from the server.</p>  |
| Usage        | <ul style="list-style-type: none"><li>• This routine provides a way for an application to respond effectively to multiple input streams. Depending on the nature of your application, the time between a request for information from the server (usually made using a call to <code>dbsqlsend</code>) and the server's response (read by calling <code>dbsqlok</code>, <code>dbresults</code>, or <code>dbnextrow</code>) may be significant. You may use this time to service other parts of your application. The <code>DBIORDESC</code> routine provides a way to obtain the I/O descriptor that a <code>DBPROCESS</code> uses to read the data stream from the server. This information may then be used with various operating system facilities (such as the UNIX <code>select</code> call) to allow the application to respond effectively to multiple input streams.</li><li>• <code>dbpoll</code> checks if a server response has arrived for any of an application's server connections (represented by <code>DBPROCESS</code> pointers). <code>dbpoll</code> is generally simpler to use than <code>DBIORDESC</code>. For this reason, and because <code>DBIORDESC</code> is non-portable, it is generally preferable to use <code>dbpoll</code>.</li><li>• The file descriptor returned by <code>DBIORDESC</code> may only be used with operating system facilities that <i>do not</i> read data from the incoming data stream. If data is read from this stream by any means other than through a DB-Library routine, communications between the front end and the server will become hopelessly scrambled.</li><li>• An application can use the DB-Library <code>DBRBUF</code> routine, in addition to the UNIX <code>select</code> function, to help determine whether any more data from the server is available for reading.</li><li>• A companion routine, <code>DBIOWDESC</code>, provides access to the file descriptor used to write data to the server.</li></ul> |
| See also     | <p><code>dbcmd</code>, <code>DBIOWDESC</code>, <code>dbnextrow</code>, <code>dbpoll</code>, <code>DBRBUF</code>, <code>dbresults</code>, <code>dbsqlok</code>, <code>dbsqlsend</code></p>  |

## DBIOWDESC

|              |   |
|--------------|---|
| Description  | (UNIX only) Provide program access to the UNIX file descriptor used by a DBPROCESS to write data to the server.   |
| Syntax       | <pre>int DBIOWDESC(dbproc)  DBPROCESS  *dbproc;</pre>   |
| Parameters   | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>  |
| Return value | An integer file descriptor used by the specified DBPROCESS to write data to the server.   |
| Usage        | <ul style="list-style-type: none"><li>• This routine provides a way for an application to effectively utilize multiple input and output streams. Depending on the nature of your application, the time interval between the initiation of an attempt to write information to the server (usually made using a call to <code>dbsqlsend</code>) and the completion of that attempt may be significant. You may use this time to service other parts of your application. The DBIOWDESC routine provides a way to obtain the I/O descriptor that a DBPROCESS uses to write the data stream to the server. This information may then be used with various operating system facilities (such as the UNIX <code>select</code> function) to allow the application to effectively utilize multiple input and output streams.</li><li>• The file descriptor returned by this routine may only be used with operating system facilities that <i>do not</i> write data to the outgoing data stream. If data is written to this stream by any means other than through a DB-Library routine, communications between the front-end and the server will become hopelessly scrambled.</li><li>• A companion routine, DBIORDESC, provides access to the file descriptor used to read data coming from the server. For some applications, another routine, <code>dbpoll</code> may be preferable to DBIORDESC.</li></ul> |
| See also     | <code>dbcmd</code> , <code>DBIORDESC</code> , <code>dbnextrow</code> , <code>dbpoll</code> , <code>dbresults</code> , <code>dbsqllok</code> , <code>dbsqlsend</code>  |



## DBISAVAIL

|              |   |
|--------------|---|
| Description  | Determine whether a DBPROCESS is available for general use.   |
| Syntax       | DBBOOL DBISAVAIL(dbproc)  |
| Parameters   | <p>DBPROCESS *dbproc;</p> <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>  |
| Return value | “TRUE” if the DBPROCESS is available for general use, otherwise “FALSE”.  |
| Usage        | This routine indicates whether the specified DBPROCESS is available for general use. When a DBPROCESS is first opened, it is marked as being available, until some use is made of it. Many DB-Library routines will automatically set the DBPROCESS to “not available,” but only dbsetavail will reset it to “available.” This facility is useful when several parts of a program are attempting to share a single DBPROCESS. |
| See also     | dbsetavail  |

## dbisopt

|             |  |
|-------------|--|
| Description | Check the status of a server or DB-Library option.   |
| Syntax      | DBBOOL dbisopt(dbproc, option, param)  |
| Parameters  | <p>DBPROCESS *dbproc;</p> <p>int option;</p> <p>char *param;</p> <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server. Unlike in the functions dbsetopt and dbclopt, <i>dbproc</i> cannot be NULL here.</p> <p>option</p> <p>The option to be checked. See Options on page 407 for the list of options.</p> |

**param**

Certain options take parameters. The DBOFFSET option, for example, takes as a parameter the SQL construct for which offsets are to be returned. Options lists those options that take parameters. If an option does not take a parameter, *param* must be NULL.

If the option you are checking takes a parameter but there can be only one instance of the option, dbisopt ignores the *param* argument. For example, dbisopt ignores the value of *param* when checking the DBBUFFER option, because row buffering can have only one setting at a time. On the other hand, the DBOFFSET option can have several settings, each with a different parameter. It may have been set twice—to look for offsets to select statements and for offsets to order by clauses. In that case, dbisopt needs the *param* argument to determine whether to check the select offset or the order by offset.

**Return value**

“TRUE” or “FALSE”.

**Usage**

- This routine checks the status of the server and DB-Library options. Although server options may be set and cleared directly through SQL, the application should instead use dbsetopt and dbcropt to set and clear options. This provides a uniform interface for setting both server and DB-Library options. It also allows the application to use the dbisopt function to check the status of an option.
- For a list of each option and its default status, see Options on page 407.

**See also**

dbcropt, dbsetopt, Options on page 407

## DBLASTROW

**Description**

Return the number of the last row in the row buffer.

**Syntax**

DBINT DBLASTROW(dbproc)

DBPROCESS \*dbproc;

**Parameters**

dbproc

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

|              |   |
|--------------|---|
| Return value | The number of the last row in the row buffer. This routine returns 0 if there is an error.  |
| Usage        | <ul style="list-style-type: none"> <li>• This macro returns the number of the last row in the row buffer. Rows are counted from the first row returned from the server, whose number is 1, and <i>not</i> from the top of the row buffer.</li> <li>• If you are not buffering rows, DBFIRSTROW, DBCURROW, and DBLASTROW will always have the same value. If you have enabled buffering by setting the DBBUFFER option, DBLASTROW will return the number of the row that is the last row in the row buffer.</li> </ul> |
| See also     | dbclrbuf, DBCURROW, DBFIRSTROW, dbgetrow, dbnextrow, dbsetopt, Options on page 407  |

## dbload\_xlate

|             |  |
|-------------|--|
| Description | Load a pair of character set translation tables.   |
| Syntax      | <pre>RETCODE dbload_xlate(dbproc, srv_charset, xlate_name,                     xlt_tosrv, xlt_todisp)</pre> <pre>DBPROCESS  *dbproc; char        *srv_charset; char        *xlt_name; DBXLATE     **xlt_tosrv; DBXLATE     **xlt_todisp;</pre>   |
| Parameters  | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>srv_charset</b><br/>A pointer to the name of the server's character set. dbload_xlate looks for a directory of this name in the <i>charsets</i> directory under the main Sybase installation directory. For example, if the server is using the iso_1 character set, dbload_xlate looks for <i>\$SYBASE/charsets/iso_1</i>.</p> <p><b>xlt_name</b><br/>A pointer to the name of the file containing the display-specific character set. dbload_xlate looks for this file in the server character set directory.</p> |

xlt\_tosrv

A pointer to a pointer to a character set translation table used to translate display-specific character strings to the server character strings. The translation table is allocated through dbload\_xlate.

xlt\_todisp

A pointer to a pointer to a character set translation table used to translate server character strings to display-specific character strings. The translation table is allocated using dbload\_xlate.

Return value

SUCCEED or FAIL.

Usage

- dbload\_xlate reads a display-specific localization file and allocates two character set translation tables: one for translations from the server's character set to the display-specific character set, and another for translations from the display-specific character set to the server's character set.
- The following code fragment illustrates the use of dbload\_xlate:

```
char          destbuf[128];
int           srcbytes_used;
DBXLATE*      xlt_todisp;
DBXLATE       *xlt_tosrv;

dbload_xlate((DBPROCESS *)NULL, "iso_1",
             "trans.xlt", &xlt_tosrv, &xlt_todisp);
printf("Original string: \n\t%s\n\n",
       TEST_STRING);
dbxlate((DBPROCESS *)NULL, TEST_STRING,
        strlen(TEST_STRING), destbuf, -1, xlt_todisp,
        &srcbytes_used);
printf("Translated to display character set: \
\t\t%s\n\n", destbuf);
dbfree_xlate((DBPROCESS *)NULL, xlt_tosrv,
             xlt_todisp);
```

See also

dbfree\_xlate, dbxlate

## dbloadsort

Description

Load a server sort order.

|              |   |
|--------------|---|
| Syntax       | DBSORTORDER *dbloadsort(dbproc)   |
|              | DBPROCESS *dbproc;  |
| Parameters   | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>  |
| Return value | A pointer to a DBSORTORDER structure on success, NULL on error.   |
| Usage        | <ul style="list-style-type: none"> <li>• dbloadsort provides information about the sort order of the server's character set. This information can be used by dbstrcmp or dbstrsort to compare two character strings.</li> <li>• dbloadsort allocates a DBSORTORDER structure to contain the server character set sort order information. The structure is freed using dbfreesort.</li> <li>• The following code fragment illustrates the use of dbloadsort: <pre> sortorder = dbloadsort(dbproc); retval = dbstrcmp(dbproc, "ABC", 3, "abc", 3,                  sortorder); printf("ABC dbstrcmp'ed with abc yields %d.\n",        retval); retval = dbstrcmp(dbproc, "abc", 3, "ABC", 3,                  sortorder); printf("abc dbstrcmp'ed with ABC yields %d.\n",        retval); dbfreesort(dbproc, sortorder); </pre> </li> </ul> |
| See also     | dbfreesort, dbstrcmp, dbstrsort   |

## dblogin

|              |  |
|--------------|--|
| Description  | Allocates a login record for use in dbopen.  |
| Syntax       | LOGINREC *dblogin()  |
| Return value | A pointer to a LOGINREC structure. dblogin returns NULL if the structure could not be allocated.                     |
| Usage        | <ul style="list-style-type: none"> <li>• This routine allocates a LOGINREC structure for use with dbopen.</li> </ul> |

- There are various routines available to supply components of the LOGINREC. The program may supply the host name, user name, user password, and application name—via DBSETLHOST, DBSETLUSER, DBSETLPWD, and DBSETAPP, respectively. It is generally only necessary for the program to supply the user password (and even this can be eliminated if the password is a null value). The other variables in the LOGINREC structure will be set to default values.
- Other components of the LOGINREC may also be changed:
  - The national language name can be set in a LOGINREC structure using DBSETLNATLANG. Call DBSETLNATLANG only if you do not wish to use the server's default national language.
  - The TDS packet size can be set in a LOGINREC using DBSETLPACKET. If not explicitly set, the TDS packet size defaults to 512 bytes. TDS is an application protocol used for the exchange of information between clients and servers.
  - The character set can be set in a LOGINREC using DBSETLCHARSET. An application needs to call DBSETLCHARSET only if it is not using ISO-8859-1 (known to the server as "iso\_1").
- When a connection attempt is made between a client and a server, there are two ways in which the connection can fail (assuming that the system is correctly configured):
  - The machine that the server is supposed to be on is running correctly and the network is running correctly.

In this case, if there is no server listening on the specified port, the machine the server is supposed to be on will signal the client, using a network error, that the connection cannot be formed. Regardless of dbsetlogintime, the connection fails.
  - The machine that the server is on is down.

In this case, the machine that the server is supposed to be on will not respond. Because "no response" is not considered to be an error, the network will not signal the client that an error has occurred. However, if dbsetlogintime has been called to set a timeout period, a timeout error will occur when the client fails to receive a response within the set period.
- Here is a program fragment that uses dblogin:

```
DBPROCESS      *dbproc;  
LOGINREC        *loginrec;
```

```
loginrec = dblogin();
DBSETLPWD(loginrec, "server_password");
DBSETLAPP(loginrec, "my_program");
dbproc = dbopen(loginrec, "my_server");
```

- Once the application has made all its dbopen calls, the LOGINREC structure is no longer necessary. The program can then call dbloginfree to free the LOGINREC structure.

See also

dbloginfree, dbopen, dbrpwclr, dbrpwset, DBSETLAPP, DBSETLCHARSET, DBSETLHOST, DBSETLNATLANG, DBSETLPACKET, DBSETLPWD, DBSETLUSER

## dbloginfree

|              |  |
|--------------|--|
| Description  | Free a login record.   |
| Syntax       | void dbloginfree(loginptr)   |
|              | LOGINREC *loginptr;  |
| Parameters   | loginptr<br>A pointer to a LOGINREC structure.   |
| Return value | None.  |
| Usage        | dblogin provides a LOGINREC structure for use with dbopen. Once the application has made all its dbopen calls, the LOGINREC structure is no longer necessary. dbloginfree frees the memory associated with the specified LOGINREC structure. |
| See also     | dblogin, dbopen  |

## dbmny4add

|             |  |
|-------------|--|
| Description | Add two DBMONEY4 values.               |
| Syntax      | RETCODE dbmny4add(dbproc, m1, m2, sum) |
|             | DBPROCESS *dbproc;                     |
|             | DBMONEY4 *m1;                          |
|             | DBMONEY4 *m2;                          |
|             | DBMONEY4 *sum;                         |

|              |   |
|--------------|---|
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><b>m1</b><br/>A pointer to a DBMONEY4 value.</p> <p><b>m2</b><br/>A pointer to a DBMONEY4 value.</p> <p><b>sum</b><br/>A pointer to a DBMONEY4 variable to hold the result of the addition.</p> |
| Return value | <p>SUCCEED or FAIL.</p> <p>dbmny4add returns FAIL in case of overflow, or if <i>m1</i>, <i>m2</i>, or <i>sum</i> is NULL.</p>   |
| Usage        | <ul style="list-style-type: none"><li>• dbmny4add adds the <i>m1</i> and <i>m2</i> DBMONEY4 values and places the result in <i>*sum</i>.</li><li>• In case of overflow, dbmny4add returns FAIL and sets <i>*sum</i> to \$0.00.</li><li>• The range of legal DBMONEY4 values is from -\$214,748.3648 to \$214,748.3647. DBMONEY4 values have a precision of one ten-thousandth of a dollar.</li></ul>  |
| See also     | <p>dbmny4sub, dbmny4mul, dbmny4divide, dbmny4minus, dbmny4add, dbmny4sub, dbmnymul, dbmnydivide, dbmnyminus</p>   |

## dbmny4cmp

|             |   |
|-------------|---|
| Description | Compare two DBMONEY4 values.  |
| Syntax      | <pre>int dbmny4cmp(dbproc, m1, m2)</pre> <pre>DBPROCESS  *dbproc;<br/>DBMONEY4   *m1;<br/>DBMONEY4   *m2;</pre> |



|              |  |
|--------------|--|
| Parameters   | <b>dbproc</b>  |
|              | A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.                |
|              | This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.     |
|              | <b>m1</b>  |
|              | A pointer to a DBMONEY4 value.   |
|              | <b>m2</b>  |
|              | A pointer to a DBMONEY4 value.   |
| Return value | If $m1 = m2$ , dbmny4cmp returns 0.  |
|              | If $m1 < m2$ , dbmny4cmp returns -1.   |
|              | If $m1 > m2$ , dbmny4cmp returns 1.  |
| Usage        | <ul style="list-style-type: none"> <li>• dbmny4cmp compares two DBMONEY4 values.</li> <li>• The range of legal DBMONEY4 values is from -\$214,748.3648 to \$214,748.3647. DBMONEY4 values have a precision of one ten-thousandth of a dollar.</li> </ul> |
| See also     | dbmnycmp   |

## dbmny4copy

|             |  |
|-------------|--|
| Description | Copy a DBMONEY4 value.   |
| Syntax      | <pre>RETCODE dbmny4copy(dbproc, src, dest)  DBPROCESS  *dbproc; DBMONEY4   *src; DBMONEY4   *dest;</pre> |

|              |  |
|--------------|--|
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><b>src</b></p> <p>A pointer to the source DBMONEY4 value.</p> <p><b>dest</b></p> <p>A pointer to the destination DBMONEY4 variable.</p> |
| Return value | <p>SUCCEED or FAIL.</p> <p>dbmny4copy returns FAIL if either <i>src</i> or <i>dest</i> is NULL.</p>  |
| Usage        | <ul style="list-style-type: none"><li>• dbmny4copy copies the <i>src</i> DBMONEY4 value to the <i>dest</i> DBMONEY4 variable.</li><li>• The range of legal DBMONEY4 values is from -\$214,748.3648 to \$214,748.3647. DBMONEY4 values have a precision of one ten-thousandth of a dollar.</li></ul>  |
| See also     | <p>dbmnycopy, dbmnyminus, dbmny4minus</p>  |

## dbmny4divide

|             |   |
|-------------|---|
| Description | Divide one DBMONEY4 value by another.   |
| Syntax      | <pre>RETCODE dbmny4divide(dbproc, m1, m2, quotient)</pre><br><pre>DBPROCESS    *dbproc;<br/>DBMONEY4     *m1;<br/>DBMONEY4     *m2;<br/>DBMONEY4     *quotient;</pre> |

|              |   |
|--------------|---|
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><b>m1</b><br/>A pointer to the DBMONEY4 value serving as dividend.</p> <p><b>m2</b><br/>A pointer to the DBMONEY4 value serving as divisor.</p> <p><b>quotient</b><br/>A pointer to a DBMONEY4 variable to hold the result of the division.</p> |
| Return value | <p>SUCCEED or FAIL.</p> <p>dbmny4divide returns FAIL in case of overflow or division by zero, or if <i>m1</i>, <i>m2</i>, or <i>quotient</i> is NULL.</p>   |
| Usage        | <ul style="list-style-type: none"> <li>• dbmny4divide divides the <i>m1</i> DBMONEY4 value by the <i>m2</i> DBMONEY4 value and places the result in <i>*quotient</i>.</li> <li>• In case of overflow or division by zero, dbmny4divide returns FAIL and sets <i>*quotient</i> to \$0.0000.</li> <li>• The range of legal DBMONEY4 values is from -\$214,748.3648 to \$214,748.3647. DBMONEY4 values have a precision of one ten-thousandth of a dollar.</li> </ul>  |
| See also     | <p>dbmny4add, dbmny4sub, dbmny4mul, dbmny4minus, dbmnyadd, dbmnysub, dbmnymul, dbmnydivide, dbmnyminus</p>  |

## dbmny4minus

|             |   |
|-------------|---|
| Description | Negate a DBMONEY4 value.  |
| Syntax      | <pre>RETCODE dbmny4minus(dbproc, src, dest)  DBPROCESS *dbproc;</pre> |

|              |   |
|--------------|---|
|              | DBMONEY4    *src;<br>DBMONEY4    *dest;   |
| Parameters   | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p>src</p> <p>A pointer to a DBMONEY4 value.</p> <p>dest</p> <p>A pointer to a DBMONEY4 variable to hold the result of the negation.</p> |
| Return value | <p>SUCCEED or FAIL.</p> <p>dbmny4minus returns FAIL in case of overflow, or if <i>src</i> or <i>dest</i> is NULL.</p>   |
| Usage        | <ul style="list-style-type: none"><li>• dbmny4minus negates the <i>src</i> DBMONEY4 value and places the result into <i>*dest</i>.</li><li>• In case of overflow, dbmny4minus returns FAIL. <i>*dest</i> is undefined in this case. An attempt to negate the maximum negative DBMONEY4 value will result in overflow.</li><li>• The range of legal DBMONEY4 values is from -\$214,748.3648 to \$214,748.3647. DBMONEY4 values have a precision of one ten-thousandth of a dollar.</li></ul>   |
| See also     | dbmnyminus, dbmnycopy, dbmny4copy   |

## dbmny4mul

|             |  |
|-------------|--|
| Description | Multiply two DBMONEY4 values.  |
| Syntax      | RETCODE dbmny4mul(dbproc, m1, m2, product)   |
|             | DBPROCESS    *dbproc;<br>DBMONEY4    *m1;<br>DBMONEY4    *m2;<br>DBMONEY4    *product; |

|              |  |
|--------------|--|
| Parameters   | <b>dbproc</b>  |
|              | A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.            |
|              | This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used. |
|              | <b>m1</b>  |
|              | A pointer to a DBMONEY4 value.   |
| Return value | <b>m2</b>  |
|              | A pointer to a DBMONEY4 value.   |
|              | <b>product</b>   |
|              | A pointer to a DBMONEY4 variable to hold the result of the multiplication.   |
|              | SUCCEED or FAIL.   |
| Usage        | dbmny4mul returns FAIL in case of overflow, or if <i>m1</i> , <i>m2</i> , or <i>product</i> is NULL.   |
|              | • dbmny4mul multiplies the <i>m1</i> DBMONEY4 value by the <i>m2</i> DBMONEY4 value and places the result in <i>*product</i> .   |
|              | • In case of overflow, dbmny4mul returns FAIL and sets <i>*product</i> to \$0.0000.  |
|              | • The range of legal DBMONEY4 values is from -\$214,748.3648 to \$214,748.3647. DBMONEY4 values have a precision of one ten-thousandth of a dollar.  |
|              |  |
| See also     | dbmny4add, dbmny4sub, dbmny4divide, dbmny4minus, dbmnyadd, dbmnysub, dbmnymul, dbmnydivide, dbmnyminus   |

## dbmny4sub

|             |   |
|-------------|---|
| Description | Subtract one DBMONEY4 value from another.     |
| Syntax      | RETCODE dbmny4sub(dbproc, m1, m2, difference) |
|             | DBPROCESS *dbproc;                            |
|             | DBMONEY4 *m1;                                 |

|              |   |
|--------------|---|
|              | DBMONEY4    *m2;<br>DBMONEY4    *difference;  |
| Parameters   | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p>m1</p> <p>A pointer to the DBMONEY4 value to be subtracted from.</p> <p>m2</p> <p>A pointer to the DBMONEY4 value to subtract.</p> <p>difference</p> <p>A pointer to a DBMONEY4 variable to hold the result of the subtraction.</p> |
| Return value | <p>SUCCEED or FAIL.</p> <p>dbmny4sub returns FAIL in case of overflow, or if <i>m1</i>, <i>m2</i>, or <i>difference</i> is NULL.</p>  |
| Usage        | <ul style="list-style-type: none"><li>• dbmny4sub subtracts the <i>m2</i> DBMONEY4 value from the <i>m1</i> DBMONEY4 value and places the result in <i>*difference</i>.</li><li>• In case of overflow, dbmny4sub returns FAIL and sets <i>*difference</i> to \$0.0000.</li><li>• The range of legal DBMONEY4 values is from -\$214,748.3648 to \$214,748.3647. DBMONEY4 values have a precision of one ten-thousandth of a dollar.</li></ul>  |
| See also     | dbmny4sub, dbmny4mul, dbmny4divide, dbmny4minus, dbmny4add, dbmnysub, dbmnymul, dbmnydivide, dbmnyminus   |

## dbmny4zero

|             |   |
|-------------|---|
| Description | Initialize a DBMONEY4 variable to \$0.0000. |
|-------------|---|

|              |   |
|--------------|---|
| Syntax       | <pre>RETCODE dbmny4zero(dbproc, mny4ptr)  DBPROCESS  *dbproc; DBMONEY4   *mny4ptr;</pre>  |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><b>mny4ptr</b><br/>A pointer to the DBMONEY4 value to initialize.</p> |
| Return value | <p>SUCCEED or FAIL.</p> <p>dbmny4zero returns FAIL if <i>mny4ptr</i> is NULL.</p>   |
| Usage        | <ul style="list-style-type: none"> <li>• dbmny4zero initializes a DBMONEY4 value to \$0.0000.</li> <li>• The range of legal DBMONEY4 values is from -\$214,748.3648 to \$214,748.3647. DBMONEY4 values have a precision of one ten-thousandth of a dollar.</li> </ul>   |
| See also     | dbmnyzero   |

## dbmnyadd

|             |  |
|-------------|--|
| Description | Add two DBMONEY values.  |
| Syntax      | <pre>RETCODE dbmnyadd(dbproc, m1, m2, sum)  DBPROCESS  *dbproc; DBMONEY    *m1; DBMONEY    *m2; DBMONEY    *sum;</pre> |

|              |  |
|--------------|--|
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><b>m1</b><br/>A pointer to a DBMONEY value.</p> <p><b>m2</b><br/>A pointer to a DBMONEY value.</p> <p><b>sum</b><br/>A pointer to a DBMONEY variable to hold the result of the addition.</p> |
| Return value | SUCCEED or FAIL.   |
| Usage        | <ul style="list-style-type: none"><li>• dbmnyadd adds the <i>m1</i> and <i>m2</i> DBMONEY values and places the result in <i>*sum</i>.</li><li>• In case of overflow, dbmnyadd returns FAIL and sets <i>*sum</i> to \$0.0000.</li><li>• The range of legal DBMONEY values is between +/- \$922,337,203,685,477.5808. DBMONEY values have a precision of one ten-thousandth of a dollar.</li><li>• dbmnyadd returns FAIL in case of overflow, or if <i>m1</i>, <i>m2</i>, or <i>sum</i> is NULL.</li></ul>  |
| See also     | dbmnysub, dbmnymul, dbmnydivide, dbmnyminus, dbmny4add, dbmny4sub, dbmny4mul, dbmny4divide, dbmny4minus  |

## dbmnycmp

|             |   |
|-------------|---|
| Description | Compare two DBMONEY values.   |
| Syntax      | <pre>int dbmnycmp(dbproc, m1, m2)</pre><br><pre>DBPROCESS  *dbproc;<br/>DBMONEY    *m1;<br/>DBMONEY    *m2;</pre> |



|              |  |
|--------------|--|
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><b>m1</b><br/>A pointer to a DBMONEY value.</p> <p><b>m2</b><br/>A pointer to a DBMONEY value.</p> |
| Return value | <p>If <math>m1 = m2</math> dbmnycmp returns 0.</p> <p>If <math>m1 &lt; m2</math> dbmnycmp returns -1.</p> <p>If <math>m1 &gt; m2</math> dbmnycmp returns 1.</p>  |
| Usage        | <ul style="list-style-type: none"> <li>• dbmnycmp compares two DBMONEY values.</li> <li>• The range of legal DBMONEY values is between +/- \$922,337,203,685,477.5808. DBMONEY values have a precision of one ten-thousandth of a dollar.</li> </ul>   |
| See also     | dbmny4cmp  |

## dbmnycopy

|             |   |
|-------------|---|
| Description | Copy a DBMONEY value.   |
| Syntax      | <pre>RETCODE dbmnycopy(dbproc, src, dest)  DBPROCESS  *dbproc; DBMONEY    *src; DBMONEY    *dest;</pre> |

|              |  |
|--------------|--|
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><b>src</b></p> <p>A pointer to the source DBMONEY value.</p> <p><b>dest</b></p> <p>A pointer to the destination DBMONEY variable.</p> |
| Return value | <p>SUCCEED or FAIL.</p> <p>dbmnycopy returns FAIL if either <i>src</i> or <i>dest</i> is NULL.</p>   |
| Usage        | <ul style="list-style-type: none"><li>• dbmnycopy copies the <i>src</i> DBMONEY value to the <i>dest</i> DBMONEY value.</li><li>• The range of legal DBMONEY values is between +/- \$922,337,203,685,477.5808. DBMONEY values have a precision of one ten-thousandth of a dollar.</li></ul>  |
| See also     | <p>dbmnycopy, dbmnyminus, dbmny4minus</p>  |

## dbmnydec

|             |   |
|-------------|---|
| Description | Decrement a DBMONEY value by one ten-thousandth of a dollar.  |
| Syntax      | <pre>RETCODE dbmnydec(dbproc, mnyptr)</pre><br><pre>DBPROCESS    *dbproc;<br/>DBMONEY      *mnyptr;</pre> |

|              |  |
|--------------|--|
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><b>mnyptr</b><br/>A pointer to the DBMONEY value to decrement.</p> |
| Return value | <p>SUCCEED or FAIL.</p> <p>dbmnydec returns FAIL in case of overflow or if <i>mnyptr</i> is NULL.</p>  |
| Usage        | <ul style="list-style-type: none"> <li>• dbmnydec decrements a DBMONEY value by one ten-thousandth of a dollar.</li> <li>• An attempt to decrement the maximum negative DBMONEY value will result in overflow. In case of overflow, dbmnydec returns FAIL. In this case, the contents of <i>*mnyptr</i> are undefined.</li> <li>• The range of legal DBMONEY values is between +/- \$922,337,203,685,477.5808. DBMONEY values have a precision of one ten-thousandth of a dollar.</li> </ul>   |
| See also     | dbmnyinc, dbmnymaxneg  |

## dbmnydivide

|             |   |
|-------------|---|
| Description | Divide one DBMONEY value by another.  |
| Syntax      | <pre> RETCODE dbmnydivide(dbproc, m1, m2, quotient)  DBPROCESS  *dbproc; DBMONEY    *m1; DBMONEY    *m2; DBMONEY    *quotient; </pre> |

|              |  |
|--------------|--|
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><b>m1</b><br/>A pointer to the DBMONEY value serving as dividend.</p> <p><b>m2</b><br/>A pointer to the DBMONEY value serving as divisor.</p> <p><b>quotient</b><br/>A pointer to a DBMONEY variable to hold the result of the division.</p> |
| Return value | <p>SUCCEED or FAIL.</p> <p>dbmnydivide returns FAIL in case of overflow or division by zero, or if <i>m1</i>, <i>m2</i>, or <i>quotient</i> is NULL.</p>   |
| Usage        | <ul style="list-style-type: none"><li>• dbmnydivide divides the <i>m1</i> DBMONEY value by the <i>m2</i> DBMONEY value and places the result in <i>*quotient</i>.</li><li>• In case of overflow or division by zero, dbmnydivide returns FAIL and sets <i>*quotient</i> to \$0.0000.</li><li>• The range of legal DBMONEY values is between +/- \$922,337,203,685,477.5808. DBMONEY values have a precision of one ten-thousandth of a dollar.</li></ul>   |
| See also     | <p>dbmnyadd, dbmnysub, dbmnymul, dbmnyminus, dbmny4add, dbmny4sub, dbmny4mul, dbmny4divide, dbmny4minus</p>  |

## dbmnydown

|             |  |
|-------------|--|
| Description | Divide a DBMONEY value by a positive integer.  |
| Syntax      | <pre>RETCODE dbmnydown(dbproc, mnyptr, divisor, remainder)</pre><br><pre>DBPROCESS  *dbproc;<br/>DBMONEY    *mnyptr;</pre> |

|              |  |
|--------------|--|
|              | <pre> int      divisor; int      *remainder; </pre>  |
| Parameters   | <p><b>dbproc</b><br/> A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><b>mnyptr</b><br/> A pointer to the DBMONEY value to divide. <i>*mnyptr</i> will also contain the result of the division.</p> <p><b>divisor</b><br/> The integer by which <i>*mnyptr</i> will be divided. <i>divisor</i> must be positive, and must be less than or equal to 65535.</p> <p><b>remainder</b><br/> A pointer to an integer variable to hold the remainder from the division, in ten-thousandths of a dollar. If <i>remainder</i> is passed as NULL, no remainder is returned.</p> |
| Return value | <p>SUCCEED or FAIL.</p> <p>dbmnydown returns FAIL if <i>mnyptr</i> is NULL, or if <i>divisor</i> is not between 1 and 65535.</p>   |
| Usage        | <ul style="list-style-type: none"> <li>• dbmnydown divides a DBMONEY value by a short integer and places the result back in the original DBMONEY variable.</li> <li>• dbmnydown places the remainder of the division into <i>*remainder</i>. <i>*remainder</i> is an integer representing the number of ten-thousandths of a dollar left after the division.</li> <li>• <i>divisor</i> must be greater than or equal to one and less than or equal to 65535.</li> <li>• The range of legal DBMONEY values is between +/- \$922,337,203,685,477.5808. DBMONEY values have a precision of one ten-thousandth of a dollar.</li> </ul>   |
| See also     | dbmnyscale, dbmnydivide, dbmny4divide  |

## dbmnyinc

|              |  |
|--------------|--|
| Description  | Increment a DBMONEY value by one ten-thousandth of a dollar.   |
| Syntax       | <pre>RETCODE dbmnyinc(dbproc, mnyptr)</pre><br><pre>DBPROCESS  *dbproc;<br/>DBMONEY    *mnyptr;</pre>  |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><b>mnyptr</b><br/>A pointer to the DBMONEY value to increment.</p> |
| Return value | SUCCEED or FAIL.<br><br>dbmnyinc returns FAIL in case of overflow or if <i>mnyptr</i> is NULL.   |
| Usage        | <ul style="list-style-type: none"><li>• dbmnyinc increments a DBMONEY value by one ten-thousandth of a dollar.</li><li>• An attempt to increment the maximum positive DBMONEY value will result in overflow. In case of overflow dbmnyinc returns FAIL. <i>*mnyptr</i> is undefined in this case.</li><li>• The range of legal DBMONEY values is between +/- \$922,337,203,685,477.5808. DBMONEY values have a precision of one ten-thousandth of a dollar.</li></ul>  |
| See also     | dbmnydec, dbmnymaxpos  |

## dbmnyinit

|             |  |
|-------------|--|
| Description | Prepare a DBMONEY value for calls to dbmnyndigit.  |
| Syntax      | <pre>RETCODE dbmnyinit(dbproc, mnyptr, trim, negative)</pre><br><pre>DBPROCESS  *dbproc;</pre> |

```
DBMONEY    *mnyptr;
int         trim;
DBBOOL      *negative;
```

## Parameters

## dbproc

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.

This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.

## mnyptr

A pointer to the DBMONEY value to be initialized. dbmnyinit changes the value of *\*mnyptr*.

## trim

The number of digits to trim from *\*mnyptr*. dbmnyinit removes digits from *\*mnyptr* by dividing it by a power of 10. The value of *trim* determines what power of 10 is used. *trim* cannot be less than 0.

## negative

A pointer to a DBBOOL variable. If *\*mnyptr* is negative, dbmnyinit makes it positive and sets *\*negative* to "true".

## Return value

SUCCEED or FAIL.

dbmnyinit returns FAIL if *mnyptr* is NULL, *negative* is NULL, or *trim* is less than 0.

## Usage

- dbmnyinit initializes a DBMONEY value for conversion to character. It eliminates unwanted precision and converts negative values to positive.
- dbmnyinit eliminates digits from a DBMONEY value by dividing by a power of 10. The integer *trim* determines what power of 10 is used. dbmnyinit modifies *\*mnyptr*, replacing the original value with the trimmed value. If *\*mnyptr* is negative, dbmnyinit makes it positive and sets *\*negative* to "true".
- dbmnyinit and dbmnyndigit are useful for writing a custom DBMONEY-to-DBCHAR conversion routine. Such a custom routine might be useful if the accuracy provided by dbconvert's DBMONEY-to-DBCHAR conversion (hundredths of a dollar) is not adequate. Also, dbconvert does not build a character string containing commas.

- dbmnyndigit returns the rightmost digit of a DBMONEY value as a DBCHAR. To get all the digits of a DBMONEY value, call dbmnyndigit repeatedly. See the dbmnyndigit reference page for more details.
- dbmnyinit is almost always used in conjunction with dbmnyndigit. Used alone, dbmnyinit can force negative DBMONEY values positive and divide DBMONEY values by a power of 10, but the real purpose of dbmnyinit is to prepare a DBMONEY value for calls to dbmnyndigit.
- The range of legal DBMONEY values is between +/- \$922,337,203,685,477.5808. DBMONEY values have a precision of one ten-thousandth of a dollar.
- The dbmnyndigit reference page contains an example that demonstrates the use of dbmnyinit.

See also

dbconvert, dbmnyndigit

## dbmnymaxneg

**Description** Return the maximum negative DBMONEY value supported.

**Syntax** RETCODE dbmnymaxneg(dbproc,dest)

DBPROCESS \*dbproc;  
DBMONEY \*dest;

**Parameters**

dbproc

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.

This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.

dest

A pointer to a DBMONEY variable.

**Return value**

SUCCEED or FAIL.

dbmnymaxneg returns FAIL if *dest* is NULL.



|          |   |
|----------|---|
| Usage    | <ul style="list-style-type: none"> <li>• <code>dbmnymaxneg</code> fills <i>*dest</i> with the maximum negative DBMONEY value supported.</li> <li>• The range of legal DBMONEY values is between +/- \$922,337,203,685,477.5808. DBMONEY values have a precision of one ten-thousandth of a dollar.</li> </ul> |
| See also | <code>dbmnymaxpos</code>  |

## dbmnymaxpos

|              |  |
|--------------|--|
| Description  | Return the maximum positive DBMONEY value supported.   |
| Syntax       | <pre>RETCODE dbmnymaxpos(dbproc, dest)</pre> <pre>DBPROCESS  *dbproc; DBMONEY    *dest;</pre>  |
| Parameters   | <p><code>dbproc</code></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><code>dest</code></p> <p>A pointer to a DBMONEY variable.</p> |
| Return value | <p>SUCCEED or FAIL.</p> <p><code>dbmnymaxpos</code> returns FAIL if <i>dest</i> is NULL.</p>   |
| Usage        | <ul style="list-style-type: none"> <li>• <code>dbmnymaxpos</code> fills <i>*dest</i> with the maximum positive DBMONEY value supported.</li> <li>• The range of legal DBMONEY values is between +/- \$922,337,203,685,477.5808. DBMONEY values have a precision of one ten-thousandth of a dollar.</li> </ul>  |
| See also     | <code>dbmnymaxneg</code>   |

## dbmnyminus

|              |   |
|--------------|---|
| Description  | Negate a DBMONEY value.   |
| Syntax       | <pre>RETCODE dbmnyminus(dbproc, src, dest)</pre><br><pre>DBPROCESS  *dbproc;<br/>DBMONEY    *src;<br/>DBMONEY    *dest;</pre>   |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><b>src</b><br/>A pointer to a DBMONEY value.</p> <p><b>dest</b><br/>A pointer to a DBMONEY variable to hold the result of the negation.</p> |
| Return value | SUCCEED or FAIL.<br><br>dbmnyminus returns FAIL in case of overflow, or if <i>src</i> or <i>dest</i> is NULL.   |
| Usage        | <ul style="list-style-type: none"><li>• dbmnyminus negates the <i>src</i> DBMONEY value and places the result into <i>*dest</i>.</li><li>• In case of overflow, dbmnyminus returns FAIL. <i>*dest</i> is undefined in this case. An attempt to negate the maximum negative DBMONEY value will result in overflow.</li><li>• The range of legal DBMONEY values is between +/- \$922,337,203,685,477.5808. DBMONEY values have a precision of one ten-thousandth of a dollar.</li></ul>   |
| See also     | dbmny4minus, dbmnycopy, dbmny4copy  |

## dbmnymul

|              |  |
|--------------|--|
| Description  | Multiply two DBMONEY values.   |
| Syntax       | <pre>RETCODE dbmnymul(dbproc, m1, m2, product)</pre> <pre>DBPROCESS  *dbproc; DBMONEY    *m1; DBMONEY    *m2; DBMONEY    *product;</pre>   |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><b>m1</b><br/>A pointer to a DBMONEY value.</p> <p><b>m2</b><br/>A pointer to a DBMONEY value.</p> <p><b>product</b><br/>A pointer to a DBMONEY variable to hold the result of the multiplication.</p> |
| Return value | <p>SUCCEED or FAIL.</p> <p>dbmnymul returns FAIL in case of overflow, or if <i>m1</i>, <i>m2</i>, or <i>product</i> is NULL.</p>   |
| Usage        | <ul style="list-style-type: none"> <li>dbmnymul multiplies the <i>m1</i> DBMONEY value by the <i>m2</i> DBMONEY value and places the result in <i>*product</i>.</li> <li>In case of overflow, dbmnymul returns FAIL and sets <i>*product</i> to \$0.0000.</li> <li>The range of legal DBMONEY values is between +/- \$922,337,203,685,477.5808. DBMONEY values have a precision of one ten-thousandth of a dollar.</li> </ul>  |
| See also     | dbmnyadd, dbmnysub, dbmnydivide, dbmnyminus, dbmny4add, dbmny4sub, dbmny4mul, dbmny4divide, dbmny4minus  |

## dbmnyndigit

|              |  |
|--------------|--|
| Description  | Return the rightmost digit of a DBMONEY value as a DBCHAR.   |
| Syntax       | <pre>RETCODE dbmnyndigit(dbproc, mnyptr, value, zero)</pre><br><pre>DBPROCESS  *dbproc;<br/>DBMONEY    *mnyptr;<br/>DBCHAR      *value;<br/>DBBOOL      *zero;</pre>   |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><b>mnyptr</b><br/>A pointer to a DBMONEY value. Each call to dbmnyndigit divides this value by 10 and places the result back into <i>*mnyptr</i>.</p> <p><b>value</b><br/>A pointer to a DBCHAR variable to fill with the character representation of the rightmost digit of the DBMONEY value.</p> <p><b>zero</b><br/>A pointer to a DBBOOL variable. Each call to dbmnyndigit divides <i>*mnyptr</i> by 10 and puts the character representation of the remainder of the division in <i>*value</i>. If the result of the division is \$0.0000, dbmnyndigit sets <i>*zero</i> to "true". Otherwise, <i>*zero</i> is set to "false". If <i>zero</i> is passed as NULL, this information is not returned.</p> |
| Return value | SUCCEED or FAIL.<br><br>dbmnyndigit returns FAIL if <i>mnyptr</i> or <i>value</i> is NULL.   |
| Usage        | <ul style="list-style-type: none"><li>dbmnyndigit returns the rightmost digit of a DBMONEY value as a DBCHAR.</li><li>dbmnyndigit divides a DBMONEY value by 10. It places the character representation of the remainder of the division in <i>*value</i>, and replaces <i>*mnyptr</i> with the result of the division. If the result of the division is \$0.0000, dbmnyndigit sets <i>*zero</i> to "true".</li></ul>  |

- To get all the digits of a DBMONEY value, call dbmnydigit repeatedly, until *\*zero* is “true”.
- dbmnyinit and dbmnydigit are useful for writing a custom DBMONEY-to-DBCHAR conversion routine. Such a custom routine might be useful if the accuracy provided by dbconvert’s DBMONEY-to-DBCHAR conversion (hundredths of a dollar) is not adequate. Also dbconvert does not build a character string containing commas.
- dbmnyinit initializes a DBMONEY value for conversion to character. It eliminates unwanted precision and converts negative values to positive. See the dbmnyinit reference page.
- The range of legal DBMONEY values is between +/- \$922,337,203,685,477.5808. DBMONEY values have a precision of one ten-thousandth of a dollar.
- This code fragment demonstrates the use of dbmnydigit and dbmnyinit:

```

/*
** This example demonstrates dbmnyinit() and
** dbmnydigit(). It is a conversion routine which
** converts a DBMONEY value to a character string.
** The conversion provided by this routine is unlike
** the conversion provided by dbconvert() in that the
** resulting character string includes commas. This
** conversion provides precision of two digits after
** the decimal point.
**
** For simplicity, the example assumes that all
** routines succeed and all parameters passed to it
** are valid.
*/

#define PRECISION      2

RETCODE      new_mnytochar(mnyptr, buf_ptr)
DBMONEY      *mnyptr;
char         *buf_ptr;
{
    DBMONEY    local_mny;
    DBBOOL     negative;
    int        bytes_written;
    DBCHAR     value;
    DBBOOL     zero;
    int        ret;

```

```
char        temp_buf[32];

/*
** Since dbmnyinit() and dbmnyndigit() modify the
** DBMONEY value passed to it, and since we do
** not want to modify the DBMONEY value passed
** to us by the user we need to make a local copy.
*/
ret = dbmnycopy((DBPROCESS *)NULL, mnyptr,
               &local_mny);
/* The value of 'ret' should be checked */

/*
** Next we need to call dbmnyinit().
**
** dbmnyinit() eliminates any unwanted precision
** from the DBMONEY value. DBMONEY values are
** stored with accuracy to four digits after the
** decimal point. For this conversion routine we
** only want accuracy to two digits after the
** decimal.
**
** Passing a value of 2 for the second parameter
** eliminates those two digits of precision we do
** not care about.
**
** dbmnyinit() also turns negative DBMONEY values
** into positive DBMONEY values. The value of
** negative is set to TRUE if dbmnyinit() turns a
** negative DBMONEY value into a positive DBMONEY
** value.
**
** NOTE: dbmnyinit() eliminates unwanted by
** precision by dividing DBMONEY values by a
** power of ten. In this conversion routine it
** divides by 100. If we pass dbmnyinit() a
** DBMONEY value of $1534.1277 the resulting
** DBMONEY value is $15.3413.
*/
negative = FALSE;
ret = dbmnyinit((DBPROCESS *)NULL, &local_mny,
               4 - PRECISION, &negative);
/* The value of 'ret' should be checked */

/*
** dbmnyndigit() extracts the rightmost digit out
```

```

** of the DBMONEY value, converts it to a
** character, places the character into the
** variable "value", and then divides the DBMONEY
** value by 10. dbmnyndigit() sets 'zero' to TRUE
** if the result of the division is $0.0000.
**
** By calling dbmnyndigit() until 'zero' is set to
** TRUE we will be returned all the digits (from
** right to left) of the DBMONEY value.
*/
zero = FALSE;
bytes_written = 0;
while( zero == FALSE )
{
    ret = dbmnyndigit((DBPROCESS *)NULL,
&local_mny, &value, &zero);
    /* The value of 'ret' should be checked. */

    /*
    ** As we are getting the digits, we want to
    ** place the decimal point and commas in the
    ** proper positions ...
    */
    temp_buf[bytes_written++] = value;

    /*
    ** If zero == TRUE we got all the digits. We
    ** do not want to call
    ** check_comma_and_decimal() since we might
    ** put a comma before the leftmost digit.
    */
    if( zero == FALSE )
    {
        /*
        ** As we are getting the digits, we want
        ** to place the decimal point and commas
        ** in the proper positions ...
        */
        check_comma_and_decimal(temp_buf,
                                &bytes_written);
    }
}

/*
** If we haven't written PRECISION bytes into the
** buffer yet, pad with zeros, write the decimal

```

```
    ** point to the buffer, and write a zero after
    ** the decimal point.
    */
    pad_with_zeros(temp_buf, &bytes_written);

    /*
    ** We've written the money value into the buffer
    ** backwards. Now we have to write it the right
    ** way.
    */
    reverse_money(buf_ptr, temp_buf, bytes_written,
                  negative);

    return(SUCCESS);
}

void check_comma_and_decimal(temp_buf,
                             bytes_written)
char *temp_buf;
int *bytes_written;
{
    static      int      comma = 0;
    static      DBBOOL   after_decimal = FALSE;

    if( after_decimal )
    {
        /*
        ** When comma is 3 it is time to write a
        ** comma. We do not care about commas until
        ** after we've written the decimal point.
        */
        comma++;
    }

    /*
    ** After we've written PRECISION bytes into the
    ** buffer, it's time to write the decimal point.
    */
    if( *bytes_written == PRECISION )
    {
        temp_buf[( *bytes_written )++] = '.';
        after_decimal = TRUE;
    }
}
```



```

/*
** When (comma == 3) that means we've written three
** digits and it's time to put a comma into the
** buffer.
*/
if( comma == 3 )
{
    temp_buf[( *bytes_written )++] = ',';
    comma = 0;          /* clear comma */
}

}

void    pad_with_zeros( temp_buf, bytes_written )
char    *temp_buf;
int     *bytes_written;
{

    /* If we haven't written PRECISION bytes into the
    ** buffer yet, pad with zeros, write the decimal
    ** point to the buffer, and write a zero after the
    ** decimal point.
    */
    while( *bytes_written < PRECISION )
    {
        temp_buf[( *bytes_written )++] = '0';
    }

    if( *bytes_written == PRECISION )
    {
        temp_buf[( *bytes_written )++] = '.';
        temp_buf[( *bytes_written )++] = '0';
    }

}

void reverse_money( char_buf, temp_buf,
                   bytes_written, negative )
char    *char_buf;
char    *temp_buf;
int     bytes_written;
DBBOOL  negative;
{

    int    i;

```

```
/*
** We've written the money value into the buffer
** backwards. Now we have to write it the right
** way. First check to see if we need to write a
** negative sign, then write the dollar sign,
** finally write the money value.
*/
i = 0;
if( negative == TRUE )
{
    char_buf[i++] = '-';
}
char_buf[i++] = '$';

while( bytes_written-- )
{
    char_buf[i++] = temp_buf[bytes_written];
}
/* Append null-terminator: */
char_buf[i] = '\0';
}
```

See also

dbconvert, dbmnyinit

## dbmnyscale

**Description** Multiply a DBMONEY value by a positive integer and add a specified amount.

**Syntax** RETCODE dbmnyscale(dbproc, mnyptr, multiplier, addend)

```
DBPROCESS *dbproc;
DBMONEY *mnyptr;
int multiplier;
int addend;
```

|              |   |
|--------------|---|
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><b>mnyptr</b><br/>A pointer to the DBMONEY value to multiply. <i>*mnyptr</i> will also contain the result of the <i>dbmnyscale</i> operation.</p> <p><b>multiplier</b><br/>The integer by which <i>*mnyptr</i> will be multiplied. <i>multiplier</i> must be positive, and must be greater than or equal to 1, and less than or equal to 65535.</p> <p><b>addend</b><br/>An integer representing the number of ten-thousandths of a dollar to add to <i>*mnyptr</i> after the multiplication.</p> |
| Return value | <p>SUCCEED or FAIL.</p> <p><i>dbmnyscale</i> returns FAIL if <i>mnyptr</i> is NULL, if overflow occurs, or if <i>multiplier</i> is not between 1 and 65535.</p>   |
| Usage        | <ul style="list-style-type: none"><li>• <i>dbmnyscale</i> multiplies a DBMONEY value by a short integer, adds <i>addend</i> ten-thousandths of a dollar, and places the result back in the original DBMONEY variable.</li><li>• <i>multiplier</i> must be greater than or equal to 1, and less than or equal to 65535.</li><li>• In case of overflow, <i>dbmnyscale</i> returns FAIL. <i>*mnyptr</i> is undefined in this case.</li><li>• The range of legal DBMONEY values is between +/- \$922,337,203,685,477.5808. DBMONEY values have a precision of one ten-thousandth of a dollar.</li></ul>   |
| See also     | <i>dbmnydown</i> , <i>dbmnymul</i> , <i>dbmny4mul</i>   |

## dbmnysub

|              |  |
|--------------|--|
| Description  | Subtract one DBMONEY value from another.   |
| Syntax       | <pre>RETCODE dbmnysub(dbproc, m1, m2, difference)</pre><br><pre>DBPROCESS    *dbproc;<br/>DBMONEY      *m1;<br/>DBMONEY      *m2;<br/>DBMONEY      *difference;</pre>  |
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><b>m1</b></p> <p>A pointer to the DBMONEY value to be subtracted from.</p> <p><b>m2</b></p> <p>A pointer to the DBMONEY value to subtract.</p> <p><b>difference</b></p> <p>A pointer to a DBMONEY variable to hold the result of the subtraction.</p> |
| Return value | <p>SUCCEED or FAIL.</p> <p>dbmnysub returns FAIL in case of overflow, or if <i>m1</i>, <i>m2</i>, or <i>difference</i> is NULL.</p>  |
| Usage        | <ul style="list-style-type: none"><li>• dbmnysub subtracts the <i>m2</i> DBMONEY value from the <i>m1</i> DBMONEY value and places the result in <i>*difference</i>.</li><li>• In case of overflow, dbmnysub returns FAIL and sets <i>difference</i> to \$0.0000.</li><li>• The range of legal DBMONEY values is between +/- \$922,337,203,685,477.5808. DBMONEY values have a precision of one ten-thousandth of a dollar.</li></ul>  |
| See also     | dbmnyadd, dbmnymul, dbmnydivide, dbmnyminus, dbmny4add, dbmny4sub, dbmny4mul, dbmny4divide, dbmny4minus  |

## dbmnyzero

|              |   |
|--------------|---|
| Description  | Initialize a DBMONEY value to \$0.0000.   |
| Syntax       | <pre>RETCODE dbmnyzero(dbproc, mnyptr)  DBPROCESS *dbproc; DBMONEY   *mnyptr;</pre>   |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> <p>This parameter may be NULL. The DBPROCESS is used as a parameter to an application's error handler. It also contains information on what language to print error messages in. If a DBPROCESS is not supplied, the default national language is used.</p> <p><b>mnyptr</b><br/>A pointer to the DBMONEY value to initialize.</p> |
| Return value | SUCCEED or FAIL.  |
|              | dbmnyzero returns FAIL if <i>mnyptr</i> is NULL.  |
| Usage        | <ul style="list-style-type: none"> <li>• dbmnyzero initializes a DBMONEY value to \$0.0000.</li> <li>• The range of legal DBMONEY values is between +/- \$922,337,203,685,477.5808. DBMONEY values have a precision of one ten-thousandth of a dollar.</li> </ul>   |
| See also     | dbmny4zero  |

## dbmonthname

|             |  |
|-------------|--|
| Description | Determine the name of a specified month in a specified language.   |
| Syntax      | <pre>char *dbmonthname(dbproc, language, monthnum,                   shortform)  DBPROCESS *dbproc; char       *language; int        monthnum; DBBOOL     shortform;</pre> |

|              |  |
|--------------|--|
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>language</b><br/>The name of the desired language.</p> <p><b>monthnum</b><br/>The number of the desired month. Month numbers range from 1 (January) to 12 (December).</p> <p><b>shortform</b><br/>A Boolean value indicating whether the long or short form of the month name is desired. If <i>shortform</i> is “true”, <i>dbmonthname</i> returns the short form of the month name; if <i>shortform</i> is “false”, <i>dbmonthname</i> returns the full month name. For example, if the month name desired is the U.S. English short form for January, “Jan” is returned.</p> <p>Short forms of month names are defined in localization files on a per-localization-file basis.</p> |
| Return value | The name of the specified month on success; a NULL pointer on error.   |
| Usage        | <ul style="list-style-type: none"><li>• <i>dbmonthname</i> returns the name of the specified month in the specified language. If no language is specified (<i>language</i> is NULL), <i>dbproc</i>’s current language is used. If both <i>language</i> and <i>dbproc</i> are NULL, DB-Library’s default language (if any) is used.</li><li>• The following code fragment illustrates the use of <i>dbmonthname</i>:<pre>for (monthnum = 1; monthnum &lt;= 12; monthnum++) printf("Month %d: %s\n", monthnum, dbmonthname((DBPROCESS *)NULL, char *)NULL, monthnum, TRUE), dbmonthname((DBPROCESS *)NULL, (char *)NULL, monthnum, FALSE));</pre></li></ul>  |
| See also     | <i>db12hour</i> , <i>dbdateorder</i> , <i>dbdayname</i> , <i>DBSETLNATLANG</i> , <i>dbsetopt</i>   |

## DBMORECMDS

|             |   |
|-------------|---|
| Description | Indicate whether there are more commands to be processed. |
|-------------|---|

|              |   |
|--------------|---|
| Syntax       | RETCODE DBMORECMDS(dbproc)  |
|              | DBPROCESS *dbproc;  |
| Parameters   | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>  |
| Return value | SUCCEED or FAIL, indicating whether there are more results from the command batch.  |
| Usage        | <ul style="list-style-type: none"><li>• The application can use this macro to determine whether there are more results to process.</li><li>• DBMORECMDS can be called after dbnextrow returns NO_MORE_ROWS. If you know that the current command is returning no rows, you can call DBMORECMDS immediately after dbresults.</li><li>• Applications rarely need this routine, because they can simply call dbresults until it returns NO_MORE_RESULTS.</li></ul> |
| See also     | DBCMDROW, dbresults, DBROWS, DBROWTYPE  |

## dbmoretext

|             |  |
|-------------|--|
| Description | Send part of a text or image value to the server.  |
| Syntax      | RETCODE dbmoretext(dbproc, size, text)   |
|             | DBPROCESS *dbproc;   |
|             | DBINT size;  |
|             | BYTE *text;  |
| Parameters  | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p>size</p> <p>The size, in bytes, of this particular part of the text or image value being sent to the server. It is an error to send more text or image bytes to the server than were specified in the call to dbwritetext.</p> |

|              |   |
|--------------|---|
|              | <p>text</p> <p>A pointer to the text or image portion to be written.</p>  |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"><li>• This routine is used in conjunction with dbwritetext to send a large SYBTEXT or SYBIMAGE value to the server in the form of a number of smaller chunks. This is particularly useful with operating systems that are unable to allocate extremely long data buffers.</li><li>• dbmoretext and dbwritetext are used in updates only, and serve to replace the Transact-SQL update statement.</li><li>• dbsqlok and dbresults must be called before the first call to dbmoretext and after the last call to dbmoretext.</li><li>• See the dbwritetext reference page.</li><li>• The DB-Library/C option DBTEXTSIZE affects the value of the server @@textsize global variable, which restricts the size of text or image values that the server returns. @@textsize has a default value of 32,768 bytes. An application that retrieves text or image values larger than 32,768 bytes will need to call dbsetopt to make @@textsize larger.</li></ul> <p>The DB-Library/C option DBTEXTLIMIT limits the size of text or image values that DB-Library/C will read.</p> |
| See also     | dbtxptr, dbtxtimestamp, dbwritetext   |

## dbmsghandle

|             |   |
|-------------|---|
| Description | Install a user function to handle server messages.  |
| Syntax      | <pre>int (*dbmsghandle(handler))()</pre> <pre>int      (*handler)();</pre>  |
| Parameters  | <p>handler</p> <p>A pointer to the user function that will be called whenever DB-Library receives an error or informational message from the server. DB-Library calls this function with eight parameters listed in Table 2-21.</p> |



**Table 2-21: Message handler parameters**

| Parameter       | Meaning   |
|-----------------|---|
| <i>dbproc</i>   | The affected DBPROCESS.   |
| <i>msgno</i>    | The current message's number (datatype DBINT). These numbers are documented in the sysmessages table.   |
| <i>msgstate</i> | The current message's error state number (datatype <i>int</i> ). These numbers provide Sybase Technical Support with information about the context of the error.  |
| <i>severity</i> | The current message's information class or error severity (datatype <i>int</i> ). These numbers are documented in the Adaptive Server Enterprise documentation.   |
| <i>msgtext</i>  | The null-terminated text of the current message (datatype <i>char *</i> ).  |
| <i>srvname</i>  | The null-terminated name of the server that generated the message (datatype <i>char *</i> ). A server's name is stored in the <i>srvname</i> column of its <i>sys.servers</i> system table. It is used in server-to-server communication; in particular, it is used when one server logs into another server to perform a remote procedure call. If the server has no name, <i>srvname</i> will be a length of 0.   |
| <i>procname</i> | The null-terminated name of the stored procedure that generated the message (datatype <i>char *</i> ). If the message was not generated by a stored procedure, <i>procname</i> will be a length of 0.   |
| <i>line</i>     | The number of the command batch or stored procedure line that generated the message (datatype <i>int</i> ). Line numbers start at 1. The line number pertains to the nesting level at which the message was generated. For instance, if a command batch executes stored procedure A, which then calls stored procedure B, and a message is generated at line 3 of B, then the value of <i>line</i> is 3.<br><i>line</i> will be 0 if there is no line number associated with the message. Circumstances that could generate messages without line numbers include a login error or a remote procedure call (performed using <i>dbrpcsend</i> ) to a stored procedure that does not exist. |

The message handler must return a value of 0 to DB-Library.

Message handlers on Windows must be declared with `CS_PUBLIC`, as shown in the following example. For portability, callback handlers on other platforms should be declared `CS_PUBLIC` as well.

The following example shows a typical message handler routine:

```
#include <sybfront.h>
#include <sybdb.h>

int CS_PUBLIC msg_handler(dbproc, msgno, msgstate,
                          severity, msgtext, srvname, procname, line)
```

```
DBPROCESS      *dbproc;
DBINT          msgno;
int            msgstate;
int            severity;
char           *msgtext;
char           *srvname;
char           *procname;
int            line;

{
    printf ("Msg %ld, Level %d, State %d\n",
            msgno, severity, msgstate);
    if (strlen(srvname) > 0)
        printf ("Server '%s', ", srvname);
    if (strlen(procname) > 0)
        printf ("Procedure '%s', ", procname);
    if (line > 0)
        printf ("Line %d", line);

    printf("\n\t%s\n", msgtext);

    return(0);
}
```

**Return value**                      A pointer to the previously installed message handler or NULL if no message handler was installed before.

**Usage**

- dbmsghandle installs a message-handler function that you supply. When DB-Library receives a server error or informational message, it will call this message handler immediately. You must install a message handler to handle server messages properly.
- If an application does not call dbmsghandle to install a message-handler function, DB-Library ignores server messages. The messages are not printed.
- If the command buffer contains just a single command and that command provokes a server message, DB-Library will call the message handler during dbsqlxexec. If the command buffer contains multiple commands (and the first command in the buffer is ok), a runtime error will not cause dbsqlxexec to fail. Instead, failure will occur with the dbresults call that processes the command causing the runtime error.
- You can “de-install” an existing message handler by calling dbmsghandle with a NULL parameter. You can also, at any time, install a new message handler. The new handler will automatically replace any existing handler.

- Refer to the `sysmessages` table for a list of server messages. In addition, the `Transact-SQL` `print` and `raiserror` commands generate server messages that `dbmsghandle` will catch.
- The routines `dbsetuserdata` and `dbgetuserdata` can be particularly useful when you need to transfer information between the message handler and the program code that triggered it. See the `dbsetuserdata` reference page for an example of how to handle deadlock in this way.
- Another routine, `dberrhandle`, installs an error handler that DB-Library calls in response to DB-Library errors.
- If the application provokes messages from DB-Library and the server simultaneously, DB-Library calls the server message handler before it calls the DB-Library error handler.
- The DB-Library/C error value `SYBESMSG` is generated in response to a server error message, but not in response to a server informational message. This means that when a server error occurs, both the server message handler and the DB-Library/C error handler are called, but when the server generates an informational message, only the server message handler is called.

If you have installed a server message handler, you may want to write your DB-Library error handler so as to suppress the printing of any `SYBESMSG` error, to avoid notifying the user about the same error twice.

- Table 2-22 provides information on when DB-Library/C calls an application's message and error handlers.

**Table 2-22: When DB-Library calls message and error handlers**

| Error or message   | Message handler called?                          | Error handler called?  |
|--|--|--|
| SQL syntax error   | Yes  | Yes (SYBESMSG).<br>(Code the handler to ignore the message.)   |
| SQL print statement  | Yes  | No.  |
| SQL raiserror  | Yes  | No.  |
| Server dies  | No   | Yes (SYBESEOF).<br>(Code your handler to exit the application.)                                      |
| Timeout from the server  | No   | Yes (SYBETIME).<br>(To wait for another timeout period, code your handler to return - INT_CONTINUE.) |
| Deadlock on query  | Yes  | No.<br>(Code your handler to test for deadlock.)   |
| Timeout on login   | No   | Yes (SYBEFCON).  |
| Login fails (dbopen)   | Yes  | Yes (SYBEPWD).<br>(Code your handler to exit the application.)                                       |
| Use database message   | Yes<br>(Code the handler to ignore the message.) | No.  |
| Incorrect use of DB-Library/C calls, such as not calling dbresults when required | No   | Yes (SYBERPND).  |
| Fatal Server error (severity greater than 16)                                    | Yes  | Yes (SYBESMSG).  |

See also

dberrhandle, dbgetuserdata, dbsetuserdata

## dbname

Description

Return the name of the current database.

|              |   |
|--------------|---|
| Syntax       | <code>char *dbname(dbproc)</code>   |
|              | <code>DBPROCESS *dbproc;</code>   |
| Parameters   | <code>dbproc</code><br>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server. |
| Return value | A pointer to the null-terminated name of the current database.  |
| Usage        | <ul style="list-style-type: none"><li>• <code>dbname</code> returns the name of the current database.</li><li>• If you need to keep track of when the database changes, use <code>dbchange</code>.</li></ul>  |
| See also     | <code>dbchange</code> , <code>dbuse</code>  |

## dbnextrow

|              |  |
|--------------|--|
| Description  | Read the next result row into the row buffer and into any program variables that are bound to column data.   |
| Syntax       | <code>STATUS dbnextrow(dbproc)</code><br><br><code>DBPROCESS *dbproc;</code>   |
| Parameters   | <code>dbproc</code><br>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.  |
| Return value | <code>dbnextrow</code> returns: <ul style="list-style-type: none"><li>• <code>REG_ROW</code> if a regular row has been read. A regular row is any row that matches the query's <code>where</code> clause.</li><li>• A <i>computeid</i> if a compute row was read. A compute row is a row that is generated by a <code>compute</code> clause. The <i>computeid</i> matches the number of the compute row that was read; the first compute row is 1, the second is 2, and so forth. A <i>computeid</i> cannot match any other of the return types for this function.</li></ul> |

## Usage

- BUF\_FULL is returned if buffering is turned on and reading the next row would cause the buffer to be exceeded. In this case, no row will have been read. To read any more rows, at least one row must first be pruned from the top of the row buffer by calling dbclrbuf.
- NO\_MORE\_ROWS if the last row in the result set has been read. If the query did not generate rows (for example, an update or insert, or a select with no match), then the first call to dbnextrow will return NO\_MORE\_ROWS. Also, dbnextrow returns this value if the query failed or if there are no pending results.
- FAIL if an abnormal event, such as a network or out-of-memory error, prevented the routine from completing successfully.
- dbnextrow reads the next row of result data, starting with the first row returned from the server. Ordinarily, the next result row is read directly from the server. If the DBBUFFER option is turned on and rows have been read out of order by calling dbgetrow, the next row is read instead from a linked list of buffered rows. When dbnextrow is called, any binding of row data to program variables (as specified with dbbind or dbaltbind) takes effect.
- If program variables are bound to columns, then new values will be written into the bound variables before dbnextrow returns.
- In regular rows, column values can be retrieved with dbdata or bound to program variables with dbbind. In compute rows, column values can be retrieved with dbadata or bound to program variables with dbaltbind.
- dbresults must return SUCCEED before an application can call dbnextrow. To determine whether a particular command is one that returns rows and needs results processing with dbnextrow, call DBROWS after dbresults.
- After calling dbresults, an application can either call dbcanquery or dbcancel to cancel the current set of results, or call dbnextrow in a loop to process the results row-by-row.
- If it chooses to process the results, an application can either:
  - Process all result rows by calling dbnextrow in a loop until it returns NO\_MORE\_ROWS. After NO\_MORE\_ROWS is returned, the application can call dbresults again to set up the next result set (if any) for processing.
  - Process some result rows by calling dbnextrow, and then cancel the remaining result rows by calling dbcancel (to cancel all results from the command batch or RPC call) or dbcanquery (to cancel only the results associated with the last dbresults call).

An application must either cancel or process all result rows.

- The typical sequence of calls is:

```
DBINT      xvariable;
DBCHAR     yvariable[10];

/* Read the query into the command buffer */
dbcmd(dbproc, "select x = 100, y = 'hello'");

/* Send the query to Adaptive Server Enterprise */
dbsqlxec(dbproc);

/* Get ready to process the query results */
dbresults(dbproc);

/* Bind column data to program variables */
dbbind(dbproc, 1, INTBIND, (DBINT) 0,
        (BYTE *) &xvariable);
dbbind(dbproc, 2, STRINGBIND, (DBINT) 0,
        yvariable);

/* Now process each row */
while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
    C-code to print or process row data
}
```

- The server can return two types of rows: regular rows containing data from columns designated by a select statement's select list, and compute rows resulting from the compute clause. To facilitate the processing of result rows from the server, `dbnextrow` returns different values according to the type of row. See the "Returns" section in this reference page for details.
- To display server result data on the default output device, you can use `dbprrow` instead of `dbnextrow`.

See also

`dbaltbind`, `dbbind`, `dbcanquery`, `dbclrbuf`, `dbgetrow`, `dbprrow`, `dbsetrow`,  
Options on page 407

## dbnpcreate

Description

Create a notification procedure.

|              |   |
|--------------|---|
| Syntax       | RETCODE dbnpcreate(dbproc)  |
|              | DBPROCESS *dbproc;  |
| Parameters   | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and the server.</p>  |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"><li>• dbnpcreate creates a notification procedure. A notification procedure is a special type of Open Server registered procedure. A notification procedure differs from a normal Open Server registered procedure in that it contains no executable statements. Notification procedures are the only type of Open Server registered procedure that a DB-Library/C application can create.</li><li>• The notification procedure name and its parameters must have been previously defined using dbnpdefine and dbregparam.</li><li>• To create a notification procedure, a DB-Library/C application must:<ul style="list-style-type: none"><li>• Define the procedure using dbnpdefine</li><li>• Describe the procedure's parameters, if any, using dbregparam</li><li>• Create the procedure using dbnpcreate</li></ul></li><li>• All DB-Library/C routines that apply to registered procedures apply to notification procedures as well. For example, dbregexec executes a registered procedure, which may or may not be a notification procedure. Likewise, dbreglist lists all registered procedures currently defined in Open Server, some of which may be notification procedures.</li><li>• Like other registered procedures, notification procedures are useful for inter-application communication and synchronization, because applications can request to be advised when a notification procedure executes.</li><li>• Notification procedures may be created only in Open Server. At this time, Adaptive Server Enterprise does not support notification procedures.</li><li>• A DB-Library/C application requests to be notified of a registered procedure's execution using dbregwatch. The application may request to be notified either synchronously or asynchronously.</li><li>• This is an example of creating a notification procedure:</li></ul> |



```
DBPROCESS    *dbproc;
DBINT        status;

/*
** Let's create a notification procedure called
** "message" which has two parameters:
**      msg      varchar(255)
**      user     idint
**/

/*
** Define the name of the notification procedure
** "message"
**/
dbnpdefine (dbproc, "message", DBNULLTERM);

/*
** The notification procedure has two parameters:
**      msg      varchar(255)
**      user     idint
** So, define these parameters. Note that
** neither of the parameters is defined with a
** default value.
**/
dbregparam (dbproc, "msg", SYBVARCHAR,
            DBNODEFAULT, NULL);
dbregparam (dbproc, "userid", SYBINT4,
            DBNODEFAULT, 4);

/* Create the notification procedure: */
status = dbnpcreate (dbproc);
if (status == FAIL)
{
    fprintf(stderr, "ERROR: Failed to create \
message!\n");
}
else
{
    fprintf(stdout, "Success in creating \
message!\n");
}
```

See also `dbreginit`, `dbregparam`, `dbregwatch`, `dbregnowatch`

## dbnpdefine

|              |   |
|--------------|---|
| Description  | Define a notification procedure.  |
| Syntax       | <pre>RETCODE dbnpdefine(dbproc, procedure_name, namelen)</pre><br><pre>DBPROCESS    *dbproc;<br/>DBCHAR       *procedure_name;<br/>DBSMALLINT   namelen;</pre>  |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and the server.</p> <p><b>procedure_name</b><br/>A pointer to the name of the notification procedure being defined.</p> <p><b>namelen</b><br/>The length of <i>procedure_name</i>, in bytes. If <i>procedure_name</i> is null-terminated, pass <i>namelen</i> as DBNULLTERM.</p>   |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"><li>• dbnpdefine defines a notification procedure. Defining a notification procedure is the first step in creating it.</li><li>• A notification procedure is a special type of Open Server registered procedure. A notification procedure differs from a normal Open Server registered procedure in that it contains no executable statements. Notification procedures are the only type of Open Server registered procedure that a DB-Library/C application can create.</li><li>• To create a notification procedure, a DB-Library/C application must:<ul style="list-style-type: none"><li>• Define the procedure using dbnpdefine</li><li>• Describe the procedure's parameters, if any, using dbregparam</li><li>• Create the procedure using dbnpcreate</li></ul></li><li>• All DB-Library/C routines that apply to registered procedures apply to notification procedures as well. For example, dbregexec executes a registered procedure, which may or may not be a notification procedure. Likewise, dbreglist lists all registered procedures currently defined in Open Server, some of which may be notification procedures.</li><li>• This is an example of defining a notification procedure:<pre>DBPROCESS    *dbproc;</pre></li></ul> |

```

DBINT          status;

/*
** Let's create a notification procedure called
** "message" which has two parameters:
**      msg      varchar(255)
**      userid   int
**/

/*
** Define the name of the notification procedure
** "message"
**/
dbnpdefine (dbproc, "message", DBNULLTERM);

/* The notification procedure has two parameters:
**      msg      varchar(255)
**      userid   int
** So, define these parameters. Note that
** neither of the parameters is defined with a
** default value.
**/
dbregparam (dbproc, "msg", SYBVARCHAR,
            DBNODEFAULT, NULL);
dbregparam (dbproc, "userid", SYBINT4,
            DBNODEFAULT, 4);

/* Create the notification procedure: */
status = dbnpcreate (dbproc);
if (status == FAIL)
{
    fprintf(stderr, "ERROR: Failed to create \
            message!\n");
}
else
{
    fprintf(stdout, "Success in creating \
            message!\n");
}

```

See also `dbregparam`, `dbnpcreate`, `dbreglist`

## dbnullbind

|   |   |
|---|---|
| Description   | Associate an indicator variable with a regular result row column.   |
| Syntax  | <pre>RETCODE dbnullbind(dbproc, column, indicator)</pre><br><pre>DBPROCESS  *dbproc;<br/>int         column;<br/>DBINT      *indicator;</pre>   |
| Parameters  | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>column</b><br/>The number of the column that is to be associated with the indicator variable.</p> <p><b>indicator</b><br/>A pointer to the indicator variable.</p>   |
| Return value  | SUCCEED or FAIL.<br><br>dbnullbind returns FAIL if <i>column</i> is invalid.  |
| Usage   | <ul style="list-style-type: none"><li>dbnullbind associates a regular result row column with an indicator variable. The indicator variable indicates whether a particular regular result row's column has been converted and copied to a program variable successfully or unsuccessfully, or whether it is null.</li><li>The indicator variable is set when regular result rows are processed using dbnextrow. The possible values are:<ul style="list-style-type: none"><li>-1 if the column is NULL.</li><li>The full length of column's data, in bytes, if <i>column</i> was bound to a program variable using dbbind, the binding did not specify any data conversions, and the bound data was truncated because the program variable was too small to hold <i>column</i>'s data.</li><li>0 if <i>column</i> was bound and copied successfully to a program variable.</li></ul></li></ul> |
| <hr/> <b>Note</b> Detection of character string truncation is implemented only for CHARBIND and VARYCHARBIND. <hr/> |   |
| See also  | dbanullbind, dbbind, dbdata, dbdatlen, dbnextrow  |

## dbnumalts

|              |   |
|--------------|---|
| Description  | Return the number of columns in a compute row.  |
| Syntax       | <pre>int dbnumalts(dbproc, computeid)</pre> <pre>DBPROCESS *dbproc; int computeid;</pre>  |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>computeid</b><br/>The ID that identifies the particular compute row of interest. A SQL select statement may have multiple compute clauses, each of which returns a separate compute row. The <i>computeid</i> corresponding to the first compute clause in a select is 1. The <i>computeid</i> is returned by dbnextrow or dbgetrow.</p> |
| Return value | The number of columns for the particular computeid. dbnumalts returns -1 if <i>computeid</i> is invalid.  |
| Usage        | <p>dbnumalts returns the number of columns in a compute row. The application can call this routine after dbresults returns SUCCEED. For example, in the following SQL statement the call dbnumalts(<i>dbproc</i>, 1) returns 3:</p> <pre>select dept, year, sales from employee order by dept, year compute avg(sales), min(sales), max(sales) by dept</pre>  |
| See also     | dbadata, dbadlen, dbaltlen, dbalttype, dbgetrow, dbnextrow, dbnumcols   |

## dbnumcols

|             |   |
|-------------|---|
| Description | Determine the number of regular columns for the current set of results. |
| Syntax      | <pre>int dbnumcols(dbproc)</pre> <pre>DBPROCESS *dbproc;</pre>          |

|              |   |
|--------------|---|
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> |
| Return value | <p>The number of columns in the current set of results. If there are no columns, <b>dbnumcols</b> returns 0.</p>  |
| Usage        | <ul style="list-style-type: none"><li>• <b>dbnumcols</b> returns the number of regular (that is, non-compute) columns in the current set of results.</li><li>• Here is a program fragment that illustrates the use of <b>dbnumcols</b>:</li></ul>                 |

```
int          column_count;
DBPROCESS   *dbproc;

/* Put the commands into the command buffer */
dbcmd(dbproc, "select name, id, type from \
sysobjects");
dbcmd(dbproc, " select name from sysobjects");

/*
** Send the commands to Adaptive Server Enterprise
and start
** execution
*/
dbsqlxexec(dbproc);

/* Process each command until there are no more */
while (dbresults(dbproc) != NO_MORE_RESULTS)
{
    column_count = dbnumcols(dbproc);
    printf("%d columns in this Adaptive Server
Enterprise \
result.\n", column_count);
    while (dbnextrow(dbproc) != NO_MORE_ROWS)
        printf("row received.\n");
}
```

|          |   |
|----------|---|
| See also | <b>dbcollen</b> , <b>dbcolname</b> , <b>dbnumalts</b> |
|----------|---|

## dbnumcompute

|              |  |
|--------------|--|
| Description  | Return the number of compute clauses in the current set of results.  |
| Syntax       | int dbnumcompute(dbproc)   |
|              | DBPROCESS *dbproc;   |
| Parameters   | dbproc<br>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.  |
| Return value | The number of compute clauses in the current set of results.   |
| Usage        | This routine returns the number of compute clauses in the current set of results. The application can call it after dbresults returns SUCCEED. For example, in the SQL statement, the call dbnumcompute( <i>dbproc</i> ) will return 2 since there are two compute clauses in the select statement: <pre> select dept, name from employee order by dept, name compute count(name) by dept compute count(name) </pre> |
| See also     | dbnumalts, dbresults   |

## DBNUMORDERS

|              |   |
|--------------|---|
| Description  | Return the number of columns specified in a Transact-SQL select statement's order by clause.  |
| Syntax       | int DBNUMORDERS(dbproc)   |
|              | DBPROCESS *dbproc;  |
| Parameters   | dbproc<br>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server. |
| Return value | The number of order by columns. If there is no order by clause, this routine returns 0. If there is an error, it returns -1.  |

|          |   |
|----------|---|
| Usage    | Once a select statement has been executed and dbresults has been called to process it, the application can call DBNUMORDERS to find out how many columns were specified in the statement's order by clause. |
| See also | dbordercol  |

## dbnumrets

|              |  |
|--------------|--|
| Description  | Determine the number of return parameter values generated by a stored procedure.   |
| Syntax       | <pre>int dbnumrets(dbproc)  DBPROCESS  *dbproc;</pre>  |
| Parameters   | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>   |
| Return value | The number of return parameter values associated with the most recently-executed stored procedure.   |
| Usage        | <ul style="list-style-type: none"><li>• dbnumrets provides the number of return parameter values returned by the most recent execute statement or remote procedure call on a stored procedure. If the number returned by dbnumrets is less than or equal to 0, then no return parameters are available.</li><li>• Transact-SQL stored procedures can return values for specified "return parameters." Changes made to the value of a return parameter inside the stored procedure are then available to the program that called the procedure. This is analogous to the "pass by reference" facility available in some programming languages.</li><li>• For a parameter to function as a return parameter, it must be declared as such within the stored procedure. The execute statement or remote procedure call that calls the stored procedure must also indicate that the parameter should function as a return parameter. In the case of a remote procedure call, it is the dbrpcparam routine that specifies whether a parameter is a return parameter.</li></ul> |



- When executing a stored procedure, the server returns any parameter values immediately after returning all other results. Therefore, the application can call `dbnumrets` only after processing the stored procedure's results by calling `dbresults`, as well as `dbnextrow` if appropriate. (Note that a stored procedure can generate several sets of results—one for each `select` it contains. Before the application can call `dbnumrets` or any other routines that process return parameters, it must call `dbresults` and `dbnextrow` as many times as necessary to process all the results.
- If the stored procedure is invoked with a remote procedure call, the return parameter values are automatically available to the application. If, on the other hand, the stored procedure is invoked with an `execute` statement, the return parameter values are available only if the command batch containing the `execute` statement uses local variables, not constants, for the return parameters. For more details on return parameters from stored procedures, see the *Adaptive Server Enterprise Reference Manual*.
- Other routines are used to retrieve return parameter values:
  - `dbretdata` returns a pointer to a parameter value.
  - `dbretlen` returns the length of a parameter value.
  - `dbretname` returns the name of a parameter value.
  - `dbrettype` returns the datatype of a parameter value.
  - `dbconvert` can be called to convert the value, if necessary.

For an example of how these routines can be used together with `dbnumrets`, see the reference page for `dbretdata`.

See also

`dbnextrow`, `dbresults`, `dbretdata`, `dbretlen`, `dbretname`, `dbrettype`, `dbrpcinit`, `dbrpcparam`

## dbopen

Description

Create and initialize a `DBPROCESS` structure.

Syntax

```
DBPROCESS *dbopen(login, server)
```

```
LOGINREC    *login;  
char        *server;
```

## Parameters

## login

A pointer to a LOGINREC structure. This pointer will be passed as an argument to dbopen. You can get one by calling dblogin.

Once the application has made all its dbopen calls, the LOGINREC structure is no longer necessary. The program can then call dbloginfree to free the LOGINREC structure.

## server

The server that you want to connect to. *server* is the alias given to the server in the interfaces file. dbopen looks up *server* in the interfaces file to get information for connecting to a server.

If *server* is NULL dbopen looks up the interfaces entry that corresponds to the value of the DSQUERY environment variable or logical name. If DSQUERY has not been explicitly set, it has a value of "SYBASE". (For information on designating an interfaces file, see the reference page for dbsetifile. See the *Open Client and Open Server Configuration Guide*.

---

**Note** On non-UNIX platforms, client applications may use a method to find server address information that is different than the UNIX *interfaces* file. Consult your *Open Client and Open Server Configuration Guide* for detailed information on how clients connect to servers.

---

## Return value

A DBPROCESS pointer if everything went well. Ordinarily, dbopen returns NULL if a DBPROCESS structure could not be created or initialized, or if your login to the server failed. When dbopen returns NULL, it generates a DB-Library error number that indicates the error. The application can access this error number through an error handler. However, if there is an unexpected communications failure during the server login process and an error handler has not been installed, the program will be aborted.

## Usage

- This routine allocates and initializes a DBPROCESS structure. This structure is the basic data structure that DB-Library uses to communicate with a server. It is the first argument in almost every DB-Library call. Besides allocating the DBPROCESS structure, this routine sets up communication with the network, logs into the server, and initializes any default options.
- Here is a program fragment that uses dbopen:

```
DBPROCESS      *dbproc;  
LOGINREC       *loginrec;  
  
loginrec = dblogin();
```

```
DBSETLPWD(loginrec, "server_password");
DBSETLAPP(loginrec, "my_program");
dbproc = dbopen(loginrec, "my_server");
```

- Once the application has logged into a server, it can change databases by calling the `dbuse` routine.

Multiple query entries in an interfaces file

- It is possible to set up an interfaces file so that if `dbopen` fails to establish a connection with a server, it attempts to establish a connection with an alternate server.
- An application can use the `dbopen` call to connect to the server MARS:

```
dbopen(loginrec, MARS);
```

An interfaces file containing an entry for MARS might look like this:

```
#
MARS
    query tcp hp-ether violet 1025
    master tcp hp-ether violet 1025
    console tcp hp-ether violet 1026
#
VENUS
    query tcp hp-ether plum 1050
    master tcp hp-ether plum 1050
    console tcp hp-ether plum 1051
#
NEPTUNE
    query tcp hp-ether mauve 1060
    master tcp hp-ether mauve 1060
    console tcp hp-ether mauve 1061
```

- The application is directed to port number 1025 on the machine “violet”. If MARS is not available, the `dbopen` call fails. If the interfaces file has multiple query entries in it for MARS, however, and the first connection attempt fails, `dbopen` will automatically attempt to connect to the next server listed. Such an interfaces file might look like this:

```
#
MARS
    query tcp hp-ether violet 1025
    query tcp hp-ether plum 1050
    query tcp hp-ether mauve 1060
    master tcp hp-ether violet 1025
    console tcp hp-ether violet 1026
```

```
#
VENUS
    query tcp hp-ether plum 1050
    master tcp hp-ether plum 1050
    console tcp hp-ether plum 1051
#
NEPTUNE
    query tcp hp-ether mauve 1060
    master tcp hp-ether mauve 1060
    console tcp hp-ether mauve 1061
```

- Note that the second query entry under MARS is identical to the query entry under VENUS, and that the third query entry is identical to the query entry under NEPTUNE. If this interfaces file is used and the application fails to connect with MARS, it will automatically attempt to connect with VENUS. If it fails to connect with VENUS, it will automatically attempt to connect with NEPTUNE. There is no limit on the number of alternate servers that may be listed under a server's interfaces file entry, but each alternate server must be listed in the same interfaces file. You can add two numbers after the server's name in the interfaces file:

```
#
MARS retries seconds
    query tcp hp-ether violet 1025
    query tcp hp-ether plum 1050
    query tcp hp-ether mauve 1060
    master tcp hp-ether violet 1025
    console tcp hp-ether violet 1026
```

*retries* represents the number of additional times to loop through the list of query entries if no connection is achieved during the first pass. *seconds* represents the amount of time, in seconds, that dbopen will wait at the top of the loop before going through the list again. These numbers are optional. If they are not included, dbopen will try to connect to each query entry only once. Looping through the list and pausing between loops is useful in case any of the candidate servers is in the process of booting. Multiple query lines can be particularly useful when alternate servers contain mirrored copies of the primary server's databases.

### Errors

The dbopen call will return NULL if any of the following errors occur. These errors can be trapped in the application's error handler (installed with dberrhandle.)

If `dbopen` is called in the entry functions of a DLL, a deadlock can arise. `dbopen` creates operating system threads and tries to synchronize them using system utilities. This synchronization conflicts with the operating system's serialization process.

**Note** The use of `SIGALARM` in a DB-Library application can cause `dbopen` to fail.

|          |  |
|----------|--|
| SYBEMEM  | Unable to allocate sufficient memory.  |
| SYBEDBPS | Maximum number of DBPROCESSes already allocated.<br><br>Note that an application can set or retrieve the maximum number of DBPROCESS structures with <code>dbsetmaxprocs</code> and <code>dbgetmaxprocs</code> . |
| SYBESOCK | Unable to open socket.   |
| SYBEINTF | Server name not found in interfaces file.  |
| SYBEUHST | Unknown host machine name.   |
| SYBECONN | Unable to connect: Adaptive Server Enterprise is unavailable or does not exist.  |
| SYBEPWD  | Login incorrect.   |
| SYBEOPIN | Could not open interfaces file.  |

See also `dbcclose`, `dbexit`, `dbinit`, `dblogin`, `dbloginfree`, `dbsetifile`, `dbuse`

## dbordercol

|             |  |
|-------------|--|
| Description | Return the id of a column appearing in the most recently executed query's order by clause.   |
| Syntax      | <code>int dbordercol(dbproc, order)</code><br><br>DBPROCESS *dbproc;<br>int order;   |
| Parameters  | <code>dbproc</code><br>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server. |

|              |   |
|--------------|---|
|              | <p>order</p> <p>The id that identifies the particular order by column of interest. The first column named within the order by clause is number 1.</p>   |
| Return value | <p>The column id (based on the column's position in the select list) for the column in the specified place in the order by clause. If the order is invalid, dbordercol returns -1.</p>  |
| Usage        | <p>This routine returns the id of the column that appears in a specified location within the order by clause of a SQL select command.</p> <p>For example, in given the SQL statement, the call <code>dbordercol(dbproc, 1)</code> will return 3 since the first column named in the order by clause refers to the third column in the query's select list:</p> <pre>select dept, name, salary from employee order by salary, name</pre> |
| See also     | <p>DBNUMORDERS</p>  |

## dbpoll

|             |  |
|-------------|--|
| Description | <p>Verifies that a server response has arrived for a DBPROCESS.</p>  |
| Syntax      | <pre>RETCODE dbpoll(dbproc, milliseconds, ready_dbproc,                return_reason)</pre> <pre>DBPROCESS  *dbproc; long        milliseconds; DBPROCESS  **ready_dbproc; int         *return_reason;</pre>  |
| Parameters  | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and server.</p> <p><i>dbproc</i> represents the DBPROCESS connection that dbpoll will check.</p> <p>If <i>dbproc</i> is passed as NULL, dbpoll will check all open DBPROCESS connections to see if a response has arrived for any of them.</p> |

**milliseconds**

The maximum number of milliseconds that dbpoll should wait for a response before returning.

If *milliseconds* is passed as 0, dbpoll returns immediately.

If *milliseconds* is passed as -1, dbpoll will not return until either a server response arrives or a system interrupt occurs.

**ready\_dbproc**

A pointer to a pointer to a DBPROCESS structure. dbpoll sets *\*ready\_dbproc* to point to the DBPROCESS for which the server response has arrived. If no response has arrived, dbpoll sets *\*ready\_dbproc* to NULL.

---

**Note** *ready\_dbproc* is not a DBPROCESS pointer. It is a pointer to a DBPROCESS pointer.

---

**return\_reason**

A pointer to an integer representing the reason dbpoll has returned. The integer will be one of the following symbolic values:

|                |  |
|----------------|--|
| DBRESULT       | A response to a server command has arrived. The application may call dbsqlok (assuming that dbsqlsend has been called) to examine the server's response.   |
| DBNOTIFICATION | A registered procedure notification has arrived. If a handler for this registered procedure has been installed using dbreghandle, dbpoll invokes this handler before it returns. If a handler for the registered procedure has not been installed and there is no default handler installed for this DBPROCESS, DB-Library raises an error when it reads the notification. |
| DBTIMEOUT      | The time indicated by the milliseconds parameter elapsed before any server response arrived.   |
| DBINTERRUPT    | An operating-system interrupt occurred before any server response arrived and before the timeout period elapsed.   |

---

**Note** This list may expand in the future, as more kinds of server responses are recognized by DB-Library/C. It is recommended that application programs be coded to handle unexpected values in *return\_reason* without error.

---

Return value

SUCCEED or FAIL.

dbpoll returns FAIL if any of the server connections it checks has died. If dbpoll returns FAIL, *ready\_dbproc* and *return\_reason* are undefined.

#### Usage

- dbpoll checks the TDS (Tabular Data Stream) buffer to see if it contains any server response not yet read by an application.
- *dbproc* represents the DBPROCESS connection that dbpoll will check. If *dbproc* is passed as NULL, dbpoll examines all open connections and returns as soon as it finds one that has an unread server response.
- If there is an unread response, dbpoll sets *\*ready\_dbproc* and *return\_reason* to reflect which DBPROCESS connection the response is for and what the response is.
- Note that *ready\_dbproc* is not a pointer to a DBPROCESS structure. It is a pointer to the address of a DBPROCESS. dbpoll sets *\*ready\_dbproc* to point to the DBPROCESS for which the server response has arrived. If no server response has arrived, dbpoll sets *\*ready\_dbproc* to NULL.
- dbpoll can be used for two purposes:
  - To allow an application to implement non-blocking reads (calls to *dbsqlok*) from the server
  - To check if a registered procedure notification has arrived for a DBPROCESS

#### Using dbpoll for non-blocking reads

- dbpoll can be used to check whether bytes are available for *dbsqlok* to read.
- Depending on the nature of an application, the time between the moment when a command is sent to the server (made using *dbsqlsend* or *dbpcsend*) and the server's response (initially read with *dbsqlok*) may be significant.



- During this time, the server is processing the command and building the result data. An application may use this time to perform other duties. When ready, the application can call `dbpoll` to check if a server response arrived while it was busy elsewhere. For an example of this usage, see the reference page for `dbsqllok`.

---

**Note** On occasion `dbpoll` may report that data is ready for `dbsqllok` to read when only the first bytes of the server response are present. When this occurs, `dbsqllok` waits for the rest of the response or until the timeout period has elapsed, just like `dbsqlexec`. In practice, however, the entire response is usually available at one time.

---

- `dbpoll` should not be used with `dbresults` or `dbnextrow`. `dbpoll` cannot determine if calls to these routines will block. This is because `dbpoll` works by checking whether or not bytes are available on a `DBPROCESS` connection, and these two routines do not always read from the network.
  - If all of the results from a command have been read, `dbresults` returns `NO_MORE_RESULTS`. In this case, `dbresults` does not block even if no bytes are available to be read.
  - If all of the rows for a result set have been read, `dbnextrow` returns `NO_MORE_ROWS`. In this case, `dbnextrow` does not block even if no bytes are available to be read.
- For non-blocking reads, alternatives to `dbpoll` are `DBRBUF` and `DBIORDESC`. These routines are specific to the UNIX-specific platform. They are not portable, so their use should be avoided whenever possible. They do, however, provide a way for application programs to integrate handling of DB-Library/C sockets with other sockets being used by an application.
  - `DBRBUF` is a UNIX-specific routine. It checks an internal DB-Library network buffer to see if a server response has already been read. `dbpoll` checks one or all connections used by an application's `DBPROCESS`s, to see if a response is ready to be read.
  - `DBIORDESC`, another UNIX-specific routine, is similar in function to `dbpoll`. `DBIORDESC` provides the socket handle used for network reads by the `DBPROCESS`. The socket handle can be used with the UNIX `select` function.

Using `dbpoll` for registered procedure notifications

- An application may have one or more DBPROCESS connections waiting for registered procedure notifications. A DBPROCESS connection will not be aware that a registered procedure notification has arrived unless it reads results from the server. If a connection is not reading results, it can use dbpoll to check if a registered procedure notification has arrived. If so, dbpoll reads the registered procedure notification stream and calls the handler for that registered procedure.
- Here is a code fragment that uses dbpoll to poll for a registered procedure notification:

```
/*
** This code fragment illustrates the use of
** dbpoll() to process an event notification.
**
** The code fragment will ask the Server to
** notify the Client when the event "shutdown"
** occurs. When the event notification is
** received from the Server, DB-Library will call
** the handler installed for that event. This
** event handler routine can then access the
** event's parameters, and take any appropriate
** action.
*/

DBINT handlerfunc();
DBINT ret;

/* First install the handler for this event */
dbreghandle(dbproc, "shutdown", handlerfunc);

/*
** Now make the asynchronous notification
** request.
*/
ret = dbregwatch(dbproc, "shutdown", DBNULLTERM,
                DBNOWAITONE);
if (ret == FAIL)
{
    fprintf(stderr, "ERROR: dbregwatch() \
                failed!!\n");
}
else if (ret == DBNOPROC)
{
    fprintf(stderr, "ERROR: procedure shutdown \
                not defined!\n");
}
```

```

    }
/*
** Since we are making use of the asynchronous
** event notification mechanism, the application
** can continue doing other work. All we have to
** do is call dbpoll() once in a while, to deal
** with the event notification when it arrives.
*/
while (1)
{
    /* Have dbpoll() block for one second */
    dbpoll(NULL, 1000, NULL, &ret);

    /*
    ** If we got the event, then get out of this
    ** loop.
    */
    if (ret == DBNOTIFICATION)
    {
        break;
    }
    /* Deal with our other tasks here */
}

```

See also `DBIORDESC`, `DBRBUF`, `dbresults`, `dbreghandle`, `dbsqlok`

## dbprhead

|              |  |
|--------------|--|
| Description  | Print the column headings for rows returned from the server.   |
| Syntax       | <pre>void dbprhead(dbproc)</pre> <p>DBPROCESS *dbproc;</p>   |
| Parameters   | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> |
| Return value | None.  |
| Usage        | <ul style="list-style-type: none"> <li>This routine displays, on the default output device and in a default format, the column headings for a set of query results. The format is compatible with the format used by <code>dbprow</code>.</li> </ul>       |

- The application can call dbprhead once dbresults returns SUCCEED.
- You can specify the maximum number of characters to be placed on one line through the DB-Library option DBPRLINELEN.
- This routine is useful for debugging.
- The routines dbsprhead, dbsprline, and dbspr1row provide an alternative to dbprhead and dbprrow. These routines print the formatted row results into a caller-supplied character buffer.

See also

dbbind, dbnextrow, dbprrow, dbresults, dbspr1row, dbsprhead, dbsprline

## dbprrow

Description

Print all the rows returned from the server.

Syntax

```
RETCODE dbprrow(dbproc)
```

```
DBPROCESS *dbproc;
```

Parameters

dbproc

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

Return value

SUCCEED or FAIL.

Usage

- This routine displays, on the default output device and in a default format, the rows for a set of query results. This routine reads and prints all the rows. It saves the trouble of calling routines such as dbbind and dbnextrow, but it prints only in a single, predetermined format.
- The application can call dbprrow once dbresults returns SUCCEED.
- When using this routine, you do not need to call dbnextrow to loop through the rows.
- You can specify the maximum number of characters to be placed on one line through the DB-Library option DBPRLINELEN.
- dbprrow is useful primarily for debugging.
- If row buffering is turned on, dbprrow buffers rows in addition to printing them out. If the buffer is full, the oldest rows are removed as necessary.

- The routines `dbsprhead`, `dbsprline`, and `dbspr1row` provide an alternative to `dbprhead` and `dbprrow`. These routines print the formatted row results into a caller-supplied character buffer.

See also

`dbbind`, `dbnextrow`, `dbprhead`, `dbresults`, `dbspr1row`, `dbsprhead`, `dbsprline`

## dbprtype

**Description** Convert a token value to a readable string.

**Syntax**

```
char *dbprtype(token)
int token;
```

**Parameters** token

The server token value (`SYBCHAR`, `SYBFLT8`, and so on) returned by `dbcoltype`, `dbaltype`, `dbrettype`, or `dbaltop`.

**Return value** A pointer to a null-terminated string that is the readable translation of the token value. The pointer points to space that is never overwritten, so it is safe to call this routine more than once in the same statement. If the token value is unknown, the routine returns a pointer to an empty string.

**Usage**

- Certain routines—`dbcoltype`, `dbaltype`, `dbrettype`, and `dbaltop`—return token values representing server datatypes or aggregate operators. `dbprtype` provides a readable string version of a token value.
- For example, `dbprtype` will take a `dbcoltype` token value representing the server binary datatype (`SYBBINARY`) and return the string “binary.”
- Table 2-23 provides a list of the token strings that `dbprtype` can return and their token value equivalents.

Table 2-23: Token values and their string equivalents

| Token string  | Token value    | Description                |
|---------------|----------------|----------------------------|
| char          | SYBCHAR        | char datatype              |
| text          | SYBTEXT        | text datatype              |
| binary        | SYBBINARY      | binary datatype            |
| image         | SYBIMAGE       | image datatype             |
| tinyint       | SYBINT1        | 1-byte integer datatype    |
| smallint      | SYBINT2        | 2-byte integer datatype    |
| int           | SYBINT4        | 4-byte integer datatype    |
| float         | SYBFLT8        | 8-byte float datatype      |
| real          | SYBREAL        | 4-byte float datatype      |
| numeric       | SYBNUMERIC     | numeric type               |
| decimal       | SYBDECIMAL     | decimal type               |
| bit           | SYBBIT         | bit datatype               |
| money         | SYBMONEY       | money datatype             |
| smallmoney    | SYBMONEY4      | 4-byte money datatype      |
| datetime      | SYBDATETIME    | datetime datatype          |
| smalldatetime | SYBDATETIME4   | 4-byte datetime datatype   |
| boundary      | SYBBOUNDARY    | boundary type              |
| sensitivity   | SYBSENSITIVITY | sensitivity type           |
| sum           | SYBAOPSUM      | sum aggregate operator     |
| avg           | SYBAOPAVG      | average aggregate operator |
| count         | SYBAOPCNT      | count aggregate operator   |
| min           | SYBAOPMIN      | minimum aggregate operator |
| max           | SYBAOPMAX      | maximum aggregate operator |

See also

dbaltop, dbalttype, dbcoltype, dbrettype, Types on page 412

# dbqual

Description

Return a pointer to a where clause suitable for use in updating the current row in a browsable table.

Syntax

char \*dbqual(dbproc, tabnum, tabname)

DBPROCESS    \*dbproc;  
int            tabnum;  
char           \*tabname;

|              |  |
|--------------|--|
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>tabnum</b><br/>The number of the table of interest, as specified in the select statement's from clause. Table numbers start at 1. If <i>tabnum</i> is -1, the <i>tablename</i> parameter will be used to identify the table.</p> <p><b>tablename</b><br/>A pointer to the null-terminated name of a table specified in the select statement's from clause. <i>tablename</i> is ignored unless <i>tabnum</i> is passed as -1.</p>  |
| Return value | <p>A pointer to a null-terminated where clause for the current row in the specified table. This buffer is dynamically allocated, and it is the application's responsibility to free it using <i>dbfreeequal</i>.</p> <p><i>dbqual</i> will return a NULL pointer if the specified table is not browsable. For a table to be "browsable," it must have a unique index and a timestamp column.</p> <p><i>dbqual</i> will also return a NULL pointer if the preceding select did not include the for browse option.</p>   |
| Usage        | <ul style="list-style-type: none"> <li><i>dbqual</i> is one of the DB-Library browse mode routines. See "Browse mode" on page 26 for a detailed discussion of browse mode.</li> <li><i>dbqual</i> provides a where clause that the application can use to update a single row in a browsable table. Columns from this row must have previously been retrieved into the application through a browse-mode select query (that is, a select that ends with the key words for browse).</li> </ul> <p>The where clause produced by <i>dbqual</i> begins with the keyword <i>where</i> and contains references to the row's unique index and timestamp column. The application simply appends the where clause to an update or delete statement; it does not need to examine it or manipulate it in any way.</p> <p>The timestamp column indicates the time that the particular row was last updated. An update on a browsable table fails if the timestamp column in the <i>dbqual</i>-generated where clause is different from the timestamp column in the table. Such a condition, which provokes Adaptive Server Enterprise error message 532, indicates that another user updated the row between the time this application selected it for browsing and the time it tried to update it. The application itself must provide the logic for handling the update failure. The following program fragment illustrates one approach:</p> <pre>/* This code fragment illustrates a technique for</pre> |

```
    ** handling the case where a browse-mode update fails
    ** because the row has already been updated
    ** by another user. In this example, we simply retrieve
    ** the entire row again, allow the user to examine and
    ** modify it, and try the update again.
    **
    ** Note that "q_dbproc" is the DBPROCESS used to query
    ** the database, and "u_dbproc" is the DBPROCESS used
    ** to update the database.
    */

/* First, find out which employee record the user
** wants to update.
*/
employee_id = which_employee();

while (1)
{
    /* Retrieve that employee record from the database.
    ** We'll assume that "empid" is a unique index,
    ** so this query will return only one row.
    */
    dbfcmd (q_dbproc, "select * from employees where \
        empid = %d for browse", employee_id);
    dbsqlexec(q_dbproc);
    dbresults(q_dbproc);
    dbnextrow(q_dbproc);

    /* Now, let the user examine or edit the employee's
    ** data, first placing the data into program
    ** variables.
    */
    extract_employee_data(q_dbproc, employee_struct);
    examine_and_edit(employee_struct, &edit_flag);

    if (edit_flag == FALSE)
    {
        /* The user didn't edit this record,

        ** so we're done.
        */
        break;
    }
    else
    {
        /* The user edited this record, so we'll use
        ** the edited data to update the
        ** corresponding row in the database.

```



Server Enterprise

```

*/
qualptr = dbqual(q_dbproc, -1, "employees");
dbcmd(u_dbproc, "update employees");
dbfcmd (u_dbproc, " set address = '%s', \
        salary = %d %s",
        employee_struct->address,
        employee_struct->salary, qualptr);
dbfreequal(qualptr);
if ((dbsqlxec(u_dbproc) == FAIL) ||
    (dbresults(u_dbproc) == FAIL))
{
    /* Our update failed. In a real program,
    ** it would be necessary to examine the
    ** messages returned from the Adaptive

    ** to determine why it failed. In this
    ** example, we'll assume that the update
    ** failed because someone else has already
    ** updated this row, thereby changing
    ** the timestamp.
    **
    ** To cope with this situation, we'll just
    ** repeat the loop, retrieving the changed
    ** row for our user to examine and edit.
    ** This will give our user the opportunity
    ** to decide whether to overwrite the
    ** change made by the other user.
    */
    continue;
}
else
{
    /* The update succeeded, so we're done. */
    break;
}
}
}

```

- dbqual can only construct where clauses for browsable tables. You can use dbtabbrowse to determine whether a table is browsable.
- dbqual is usually called after dbnextrow.
- For a complete example that uses dbqual to perform a browse mode update, see the sample programs included with DB-Library.

See also

dbcolbrowse, dbcolsource, dbfreequal, dbtabbrowse, dbtabcount, dbtabname, dbtabsource, dbtsnewlen, dbtsnewval, dbtsput

## DBRBUF

|              |  |
|--------------|--|
| Description  | (UNIX only) Determine whether the DB-Library network buffer contains any unread bytes.   |
| Syntax       | <pre>DBBOOL DBRBUF(dbproc)</pre><br><pre>DBPROCESS  *dbproc;</pre>   |
| Parameters   | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>   |
| Return value | <p>“TRUE” (bytes remain in buffer) or “FALSE” (no bytes in buffer).</p> <p>Note that DBRBUF actually returns “TRUE” both when there are bytes available in the read buffer, and when no more results are available to be processed.</p> <p>This is because the purpose of DBRBUF is to tell an application when it can read and be assured that it will not hang. If DBRBUF did not return “TRUE” in the case of no more results, then applications that loop while DBRBUF returns “FALSE” could loop indefinitely, if all results had already been processed.</p>   |
| Usage        | <ul style="list-style-type: none"><li>• This routine lets the application know if the DB-Library network buffer contains any bytes yet unread.</li><li>• DBRBUF is ordinarily used in conjunction with dbsqlok and DBIORDESC.</li><li>• dbpoll, a DB-Library/C routine which checks if a server response has arrived for any DBPROCESS, may replace DBRBUF. Since the UNIX-specific routines DBRBUF and DBIORDESC are non-portable, their use should be avoided whenever possible. They do, however, provide a way for application programs to integrate handling of DB-Library/C sockets with other sockets being used by an application.</li><li>• An application uses these routines to manage multiple input data streams. To manage these streams efficiently, an application that uses dbsqlok should check whether any bytes remain either in the network buffer or in the network itself before calling dbresults.</li><li>• To test whether bytes remain in the network buffer, the application can call DBRBUF. To test whether bytes remain in the network itself, the application can either call the UNIX select and DBIORDESC, or call dbpoll.</li></ul> |
| See also     | DBIORDESC, dbpoll, dbsqlok, dbresults  |

## dbreadpage

|              |   |
|--------------|---|
| Description  | Read a page of binary data from the server.   |
| Syntax       | <pre>DBINT dbreadpage(dbproc, dbname, pageno, buf)</pre> <pre>DBPROCESS *dbproc;<br/>char *dbname;<br/>DBINT pageno;<br/>BYTE buf[];</pre>  |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>dbname</b><br/>The name of the database of interest.</p> <p><b>pageno</b><br/>The number of the database page to be read.</p> <p><b>buf</b><br/>A pointer to a buffer to hold the received page data. Adaptive Server Enterprise pages are currently 2048 bytes long.</p>                          |
| Return value | The number of bytes read from the server. If the operation was unsuccessful, dbreadpage returns -1.   |
| Usage        | <ul style="list-style-type: none"><li>dbreadpage reads a page of binary data from the server. This routine is primarily useful for examining and repairing damaged database pages. After calling dbreadpage, the DBPROCESS may contain some error or informational messages from the server. These messages may be accessed through a user-supplied message handler.</li><li>dbreadpage alters the contents of the DBPROCESS command buffer.</li></ul> <hr/> <p><b>Warning!</b> Use this routine only if you are absolutely sure you know what you are doing!</p> <hr/> |
| See also     | dbmsghandle, dbwritepage  |

# dbreadtext

|              |   |
|--------------|---|
| Description  | Read part of a text or image value from the server.   |
| Syntax       | STATUS dbreadtext(dbproc, buf, bufsize)   |
|              | <pre>DBPROCESS  *dbproc; void        *buf; DBINT      bufsize;</pre>  |
| Parameters   | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p>buf</p> <p>A pointer to a caller-allocated buffer that will contain the chunk of text or image data.</p> <p>bufsize</p> <p>The size of the caller's buffer, in bytes.</p> |
| Return value | The following table lists the return values for dbreadtext:   |

| dbreadtext<br>returns | To indicate   |
|-----------------------|---|
| >0                    | The number of bytes placed into the caller's buffer         |
| 0                     | The end of a row  |
| -1                    | An error occurred, such as a network or out of memory error |
| NO_MORE_ROWS          | All rows read   |

|       |  |
|-------|--|
| Usage | <ul style="list-style-type: none"><li>dbreadtext reads a large SYBTEXT or SYBIMAGE value from the server in the form of a number of smaller chunks. This is particularly useful with operating systems that are unable to allocate extremely long data buffers.</li><li>To read successive chunks of the same SYBTEXT or SYBIMAGE value, call dbreadtext until it returns 0 (end of row).</li><li>Use dbreadtext in place of dbnextrow to read SYBTEXT and SYBIMAGE values.</li><li>dbreadtext can process the results of Transact-SQL queries if those queries return only one column and that column contains either text or image data. The Transact-SQL readtext command returns results of this type.</li></ul> |
|-------|--|

- The DB-Library/C option DBTEXTSIZE affects the value of the server @@*textsize* global variable, which restricts the size of text or image values that the server returns. @@*textsize* has a default value of 32,768 bytes. An application that retrieves text or image values larger than 32,768 bytes will need to call dbsetopt to make @@*textsize* larger.

The DB-Library/C option DBTEXTLIMIT limits the size of text or image values that DB-Library/C will read. DB-Library/C will throw away any text that exceeds the limit.

- This code fragment demonstrates the use of dbreadtext:

```
DBPROCESS      *dbproc;
long           bytes;
RETCODE        ret;
char           buf[BUFSIZE + 1];
/*
** Install message and error handlers...
** Log in to server...
** Send a "use database" command...
**/
/* Select a text column: */
dbfcmd(dbproc, "select textcolumn from bigtable");
dbsqlexec(dbproc);

/* Process the results: */
while( (ret = dbresults(dbproc)) !=
        NO_MORE_RESULTS )
{
    if( ret == FAIL )
    {
        /* dbresults() failed */
    }
    while( (bytes =
dbreadtext(dbproc,
            (void *)buf, BUFSIZE)) != NO_MORE_ROWS )
    {
        if( bytes == -1 )
        {
            /* dbreadtext() failed */
        }
        else if( bytes == 0 )
        {
            /* We've reached the end of a row*/
            printf("End of Row!\n\n");
        }
        else

```

```
        {
            /*
             ** 'bytes' bytes have been placed
             ** into our buffer.
             ** Print them:
             */
            buf[bytes] = '\0';
            printf("%s\n", buf);
        }
    }
```

See also `dbmoretext`, `dbnextrow`, `dbwritetext`

## dbrecftos

**Description** Record all SQL commands sent from the application to the server.

**Syntax** `void dbrecftos(filename)`

`char *filename;`

**Parameters** `filename`  
A pointer to a null-terminated character string to be used as the basis for naming SQL session files.

**Return value** None.

**Usage**

- `dbrecftos` causes all SQL commands sent from the front-end application program to the server to be recorded in a human-readable file. This SQL session information is useful for debugging purposes.
- DB-Library creates one SQL session file for each call to `dbopen` that occurs after `dbrecftos` is called. Files are named *filename.n*, where *filename* is the name specified in the call to `dbrecftos` and *n* is an integer, starting with 0.

For example, if *filename* is “foo,” the first file created is named *foo.0*, the next *foo.1*, and so forth.

See also `dbopen`

## dbrecvpassthru

|              |   |
|--------------|---|
| Description  | Receive a TDS packet from a server.   |
| Syntax       | <pre>RETCODE dbrecvpassthru(dbproc, recv_bufp)  DBPROCESS  *dbproc; DBVOIDPTR  *recv_bufp;</pre>  |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and the server.</p> <p><b>recv_bufp</b><br/>A pointer to a variable that dbrecvpassthru fills with the address of a buffer containing the TDS packet most recently received by this DBPROCESS connection. The application is not responsible for allocating this buffer.</p>   |
| Return value | DB_PASSTHRU_MORE, DB_PASSTHRU_EOM, or FAIL.   |
| Usage        | <ul style="list-style-type: none"> <li>dbrecvpassthru receives a TDS (Tabular Data Stream) packet from a server.</li> <li>TDS is an application protocol used for the transfer of requests and request results between clients and servers. Under ordinary circumstances, a DB-Library/C application does not have to deal directly with TDS, because DB-Library/C manages the data stream.</li> <li>dbrecvpassthru and dbsendpassthru are useful in gateway applications. When an application serves as the intermediary between two servers, it can use these routines to pass the TDS stream from one server to the other, eliminating the process of interpreting the information and re-encoding it.</li> <li>dbrecvpassthru reads a packet of bytes from the server connection identified by <i>dbproc</i> and sets <i>*recv_bufp</i> to point to the buffer containing the bytes.</li> <li>A packet has a default size of 512 bytes. An application can change its packet size using DBSETLPACKET. See the dbgetpacket and DBSETLPACKET reference pages.</li> <li>dbrecvpassthru returns DB_PASSTHRU_EOM if the TDS packet has been marked by the server as EOM (End Of Message). If the TDS packet is not the last in the stream, dbrecvpassthru returns DB_PASSTHRU_MORE.</li> </ul> |

- A DBPROCESS connection which is used for a dbrecvpassthru operation cannot be used for any other DB-Library/C function until DB\_PASSTHRU\_EOM has been received.
- This is a code fragment using dbrecvpassthru:

```
/*
** The following code fragment illustrates the
** use of dbrecvpassthru() in an Open Server
** gateway application. It will continually get
** packets from a remote server, and pass them
** through to the client.
**
** The routine srv_sendpassthru() is the Open
** Server counterpart required to complete
** this passthru operation.
*/
DBPROCESS    *dbproc;
SRV_PROC     *srvproc;
int          ret;
BYTE         *packet;

while(1)
{
    /* Get a TDS packet from the remote server */
    ret = dbrecvpassthru(dbproc, &packet);

    if( ret == FAIL )
    {
        fprintf(stderr, "ERROR - dbrecvpassthru\
            failed in handle_results.\n");
        exit();
    }
    /* Now send the packet to the client */
    if( srv_sendpassthru(srvproc, packet,
        (int *)NULL) == FAIL )
    {
        fprintf(stderr, "ERROR - srv_sendpassthru \
            failed in handle_results.\n");
        exit();
    }
}
/*
** We've sent the packet, so let's see if
** there's any more.
*/
if( ret == DB_PASSTHRU_MORE )
    continue;
else
```



```
break;
}
```

See also `dbsendpassthru`

## dbregdrop

**Description** Drop a registered procedure.

**Syntax** `RETCODE dbregdrop(dbproc, procedure_name, namelen)`

```
DBPROCESS *dbproc;
DBCHAR    *procedure_name;
DBSMALLINT namelen;
```

**Parameters** `dbproc`

A pointer to the `DBPROCESS` structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and the server.

`procedure_name`

A pointer to the name of the registered procedure that the `DBPROCESS` connection wishes to drop.

`namelen`

The length of *procedure\_name*, in bytes. If *procedure\_name* is null-terminated, pass *namelen* as `DBNULLTERM`.

**Return value** `SUCCEED`, `DBNOPROC`, or `FAIL`.

**Usage**

- `dbregdrop` drops a registered procedure from Open Server. Because a notification procedure is simply a special type of registered procedure, a notification procedure may also be dropped using `dbregdrop`.
- A `DBPROCESS` connection can drop any registered procedure defined in Open Server, including procedures created by other `DBPROCESS` connections and procedures created by other applications. Any mechanism to protect registered procedures must be embodied in the server application.
- If the procedure referenced by *procedure\_name* is not defined in Open Server, `dbregdrop` returns `DBNOPROC`. An application can use `dbreglist` to obtain a list of registered procedures currently defined in Open Server.
- This is a code fragment that uses `dbregdrop`:

```
/*
** The following code fragment illustrates
** dropping a registered procedure.
*/
DBPROCESS    *dbproc;
RETCODE      ret;
char         *procname;

procname = "some_event";
ret = dbregdrop(dbproc, procname, DBNULLTERM);
if (ret == FAIL)
{
    fprintf(stderr, "ERROR: dbregdrop() \
                failed!!\n");
}
else if (ret == DBNOPROC)
{
    fprintf(stderr, "ERROR: procedure %s was not\
                registered!\n", procname);
}
```

See also `dbnpcreate`, `dbreglist`

## dbregexec

Description                      Execute a registered procedure.

Syntax                           RETCODE dbregexec(dbproc, options)

```
DBPROCESS    *dbproc;
DBUSMALLINT   options;
```

Parameters                      dbproc

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and the server.

## options

A 2-byte bitmask, either DBNOTIFYALL or DBNOTIFYNEXT.

If options is DBNOTIFYALL, Open Server will notify all DBPROCESSes watching for the execution of this registered procedure.

If options is DBNOTIFYNEXT, Open Server will notify only the DBPROCESS that has been watching the longest.

## Return value

SUCCEED or FAIL.

## Usage

- dbregexec completes the process of executing a registered procedure. Because a notification procedure is simply a special type of registered procedure, a notification procedure may also be executed using dbregexec.
- The procedure name and its parameters must have been previously defined using dbreginit and dbregparam.
- To execute a registered procedure, a DB-Library/C application must:
  - Initiate the call using dbreginit.
  - Describe the procedure's parameters, if any, using dbregparam.
  - Execute the procedure using dbregexec.
- An application cannot execute a registered procedure that is not defined in Open Server. dbreglist returns a list of registered procedures that are currently defined.
- Registered procedures are useful for inter-application communication and synchronization, because applications can request to be advised when a registered procedure executes.
- Registered procedures may be created only in Open Server. At this time, Adaptive Server Enterprise does not support registered procedures. An application can use dbnproc, dbregparam, and dbnproc to create a registered procedure.
- A DB-Library/C application requests to be notified of a registered procedure's execution using dbregwatch. The application may request to be notified either synchronously or asynchronously.
- This is an example of executing a registered procedure:

```
DBPROCESS    *dbproc;
DBINT        newprice = 55;
DBINT        status;
/*
** Initiate execution of the registered procedure
** "price_change"
```

```
*/
dbreginit (dbproc, "price_change", DBNULLTERM);
/*
** The registered procedure has two parameters:
**     name          varchar(255)
**     newprice      int
** So pass these parameters to the registered
** procedure.
*/
dbregparam (dbproc, "name", SYBVARCHAR, NULL,
            "sybase");
dbregparam (dbproc, "newprice", SYBINT4, 4,
            &newprice);
/* Execute the registered procedure: */
status = dbregexec (dbproc, DBNOTIFYALL);
if (status == FAIL)
{
    fprintf(stderr, "ERROR: Failed to execute \
price_change!\n");
}
else if (status == DBNOPROC)
{
    fprintf(stderr, "ERROR: Price_change does \
not exist!\n");
}
else
{
    fprintf(stdout, "Success in executing \
price_change!\n");
}
```

See also                   dbreginit, dbregparam, dbregwatch, dbregnowatch

## dbreghandle

|             |  |
|-------------|--|
| Description | Install a handler routine for a registered procedure notification. |
| Syntax      | RETCODE dbreghandle(dbproc, procedure_name, namelen,<br>handler)   |
| DBPROCESS   | *dbproc;   |
| DBCHAR      | *procedure_name;   |

|              |  |
|--------------|--|
|              | DBSMALLINT    namelen;<br>INTFUNCPTR    handler;   |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and the server.</p> <p><b>procedure_name</b><br/>A pointer to the name of the registered procedure for which the handler is being installed.</p> <p>If <i>procedure_name</i> is passed as NULL, the handler is installed as a default handler. The default handler will be called for all registered procedure notifications read by this DBPROCESS connection for which no other handler has been installed.</p> <p><b>namelen</b><br/>The length of <i>procedure_name</i>, in bytes. If <i>procedure_name</i> is null-terminated, pass <i>namelen</i> as DBNULLTERM.</p> <p><b>handler</b><br/>A pointer to the function to be called by DB-Library/C when the registered procedure notification is read.</p> <p>If <i>handler</i> is passed as NULL, any handler previously installed for the registered procedure is uninstalled.</p> |
| Return value | SUCCEED or FAIL.   |
| Usage        | <ul style="list-style-type: none"> <li>dbreghandle installs a user-supplied handler routine to be called by DB-Library/C when a DBPROCESS connection reads an asynchronous notification that a registered procedure has been executed.</li> </ul> <p>Because a notification procedure is simply a special type of registered procedure, a handler for a notification procedure may also be installed using dbreghandle.</p> <ul style="list-style-type: none"> <li>An application receives an asynchronous notification only if it has previously called dbregwatch with <i>options</i> passed as DBNOWAITONE or DBNOWAITALL. This call tells Open Server that the application is interested in the execution of the registered procedure, that it will receive the notification asynchronously, and that it will read the notification through a particular DBPROCESS connection.</li> <li>If no handler is installed for a notification, DB-Library/C will raise an error when the DBPROCESS connection reads the notification.</li> </ul>   |

- Either *procedure\_name* or *handler* may be NULL:
  - If both *procedure\_name* and *handler* are supplied, dbreghandle installs the handler specified by *handler* for the registered procedure specified by *procedure\_name*.
  - If *procedure\_name* is NULL and *handler* is NULL, dbreghandle uninstalls all handlers for this DBPROCESS connection.
  - If *procedure\_name* is NULL but *handler* is supplied, dbreghandle installs the handler specified by *handler* as a “default” handler for this DBPROCESS connection. This default handler will be called whenever the DBPROCESS connection reads a registered procedure notification for which no other handler has been installed.
  - If *procedure\_name* is supplied but *handler* is NULL, dbreghandle uninstalls any handler previously installed for this registered procedure. If a default handler has been installed for this DBPROCESS connection, it remains in effect and will be called if a *procedure\_name* notification is read.
- The same handler may be used by several DBPROCESS connections, but it must be installed for each one by a separate call to dbreghandle. Because of the possibility of a single notification handler being called when different DBPROCESSes read notifications, all handlers should be written to be re-entrant.
- A single DBPROCESS connection may be watching for several registered procedures to execute. This connection may have different handlers installed to process the various notifications it may read. Each handler must be installed by a separate call to dbreghandle.
- A DBPROCESS connection may be idle, sending commands, reading results, or idle with results pending when a registered procedure notification arrives.
  - If the DBPROCESS connection is idle, it is necessary for the application to call dbpoll to allow the connection to read the notification. If a handler for the notification has been installed, it will be called before dbpoll returns.
  - If the DBPROCESS connection is sending commands, the notification is read and the notification handler called during dbsqlxexec or dbsqlok. After the notification handler returns, flow of control continues normally.

- If the DBPROCESS connection is reading results, the notification is read and the notification handler called either in `dbresults` or `dbnextrow`. After the notification handler returns, flow of control continues normally.
- If the DBPROCESS connection is idle with results pending, the notification is not read until all results in the stream up to the notification have been read and processed by the connection.
- Because a notification may be read while a DBPROCESS connection is in any of several different states, the actions that a notification handler may take are restricted. A notification handler may not use an existing DBPROCESS to send a query to the server, process the results of a query, or call `dbcancel` or `dbcancelquery`. A notification handler may, however, open a new DBPROCESS and use this new DBPROCESS to send queries and process results within the handler.
- A notification handler can read the arguments passed to the registered procedure upon execution. To do this, the handler can use the DB-Library/C routines `dbnumrets`, `dbrettype`, `dbretlen`, `dbretname`, and `dbretdata`.
- All notification handlers are called by DB-Library/C with the following parameters:
  - *dbproc*, a pointer to the DBPROCESS connection that has been watching for the notification
  - *procedure\_name*, a pointer to the name of the registered procedure that has been executed
  - *reserved1*, a DBUSMALLINT parameter reserved for future use
  - *reserved2*, a DBUSMALLINT parameter reserved for future use
- A notification handler must return `INT_CONTINUE` to indicate normal completion, or `INT_EXIT` to instruct DB-Library/C to abort the application and return control to the operating system.
- Notification handlers on the Windows platform must be declared with `CS_PUBLIC`, as shown in the example below. For portability, callback handlers on other platforms should be declared `CS_PUBLIC` as well.
- This is an example of a notification handler:

```
DBINT CS_PUBLIC my_procedure_handler(dbproc,
                                     procedure_name, reserved1, reserved2)
/* The client connection */
DBPROCESS      *dbproc;
```

```
/* A null-terminated string */
DBCHAR      *procedure_name;
/* Reserved for future use */
DBUSMALLINT reserved1;
/* Reserved for future use */
DBUSMALLINT reserved2;

{
    int      i, type;
    DBINT    len;
    char     *name;
    BYTE     *data;
    int      params;

    /*
     ** Find out how many parameters this
     ** procedure received.
     */
    params = dbnumrets(dbproc);

    i = 0;    /* Initialize counter */

    /* Now process each parameter in turn */
    while(i++ < params)
    {
        /* Get the parameter's datatype */
        type = dbrettype(dbproc, i);

        /* Get the parameter's length */
        len = dbretlen(dbproc, i);

        /* Get the parameter's name */
        name = dbretname(dbproc, i);

        /* Get a pointer to the parameter */
        data = dbretdata(dbproc, i);

        /* Process the parameter here */
    }
    return(INT_CONTINUE);
}
```

See also

dbregwatch, dbregnowatch, dbregparam, dbregexec

## dbreginit

Description

Initiate execution of a registered procedure.



|              |   |
|--------------|---|
| Syntax       | <pre>RETCODE dbreginit(dbproc, procedure_name, namelen)  DBPROCESS  *dbproc; DBCHAR     *procedure_name; DBSMALLINT namelen;</pre>  |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and server.</p> <p><b>procedure_name</b><br/>A pointer to the name of the registered procedure being executed.</p> <p><b>namelen</b><br/>The length of <i>procedure_name</i>, in bytes. If <i>procedure_name</i> is null-terminated, pass <i>namelen</i> as DBNULLTERM.</p>  |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"> <li>• dbreginit initiates the execution of a registered procedure. Because a notification procedure is simply a special type of registered procedure, execution of a notification procedure may also be initiated using dbreginit.</li> <li>• To execute a registered procedure, a DB-Library/C application must:             <ul style="list-style-type: none"> <li>• Initiate the call using dbreginit</li> <li>• Pass the procedure's parameters, if any, using dbregparam</li> <li>• Execute the procedure using dbregexec</li> </ul> </li> <li>• This is an example of executing a registered procedure:             <pre>DBPROCESS  *dbproc; DBINT      newprice = 55; DBINT      status;  /* ** Initiate execution of the registered procedure ** "price_change". ** dbreginit (dbproc, "price_change", DBNULLTERM);  /* ** The registered procedure has two parameters: **      name      varchar(255) **      newprice  int ** So pass these parameters to the registered ** procedure.</pre> </li> </ul> |

```
*/
dbregparam (dbproc, "name", SYBVARCHAR, NULL,
            "sybase");
dbregparam (dbproc, "newprice", SYBINT4, 4, 4,
            &newprice);
/* Execute the registered procedure: */
status = dbregexec (dbproc, DBNOTIFYALL);
if (status == FAIL)
{
    fprintf(stderr, "ERROR: Failed to execute \
price_change!\n");
}
else if (status == DBNOPROC)
{
    fprintf(stderr, "ERROR: Price_change does \
not exist!\n");
}
else
{
    fprintf(stdout, "Success in executing \
price_change!\n");
}
```

See also [dbregparam](#), [dbregexec](#), [dbregwatch](#), [dbreglist](#), [dbregwatchlist](#)

## dbreglist

|              |   |
|--------------|---|
| Description  | Return a list of registered procedures currently defined in Open Server.  |
| Syntax       | RETCODE dbreglist(dbproc)   |
|              | DBPROCESS *dbproc;  |
| Parameters   | dbproc<br>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and the server. |
| Return value | SUCCEED or FAIL.  |

## Usage

- dbreglist returns a list of registered procedures currently defined in Open Server. Because a notification procedure is simply a special type of registered procedure, notification procedures will be included in the list of registered procedures.
- The list of registered procedures is returned as rows that an application must explicitly process after calling dbreglist. Each row represents the name of a single registered procedure defined in Open Server. A row contains a single column of type SYBVARCHAR.
- The following code fragment illustrates how dbreglist might be used in an application:

```
DBPROCESS      *dbproc;
DBCHAR          *procedurename;
DBINT           ret;

/* request the list of procedures */
if( (ret = dbreglist(dbproc)) == FAIL)
{
    /* Handle failure here */
}
dbresults(dbproc);
while( dbnextrow(dbproc) != NO_MORE_ROWS )
{
    procedurename = (DBCHAR *)dbdata(dbproc, 1);
    procedurename[dbdatlen(dbproc, 1)] = '\0';

    fprintf(stdout, "The procedure '%s' is \
defined.\n", procedurename);
}
/* All done */
```

## See also

dbregwatchlist, dbregwatch

## dbregnowatch

## Description

Cancel a request to be notified when a registered procedure executes.

## Syntax

```
RETCODE dbregnowatch(dbproc, procedure_name,
                    namelen)
```

```
DBPROCESS      *dbproc;
DBCHAR          *procedure_name;
DBSMALLINT      namelen;
```

|              |  |
|--------------|--|
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and the server.</p> <p><b>procedure_name</b><br/>A pointer to the name of the registered procedure that the DBPROCESS connection is no longer interested in.</p> <p><b>namelen</b><br/>The length of <i>procedure_name</i>, in bytes. If <i>procedure_name</i> is null-terminated, pass <i>namelen</i> as DBNULLTERM.</p>   |
| Return value | SUCCEED, DBNOPROC, or FAIL.  |
| Usage        | <ul style="list-style-type: none"><li>• dbregnowatch cancels a DBPROCESS connection's request to be notified when a registered procedure executes. Because a notification procedure is simply a special type of registered procedure, dbregnowatch also cancels a DBPROCESS connection's request to be notified when a notification procedure executes.</li><li>• It is meaningful to call dbregnowatch only if the DBPROCESS connection has previously requested an asynchronous notification using dbregwatch.</li><li>• If the procedure referenced by <i>procedure_name</i> is not defined in Open Server, dbregnowatch returns DBNOPROC. An application can obtain a list of procedures currently registered in Open Server using dbreglist.</li><li>• An application can obtain a list of registered procedures it is watching for through dbregwatchlist.</li><li>• This is an example of canceling a request to be notified:</li></ul> |

```
DBPROCESS    *dbproc;
DBINT        ret;

/*
** Inform the server that we no longer wish to
** be notified when "price_change" executes:
*/
ret = dbregnowatch (dbproc, "price_change",
                   DBNULLTERM);
if (ret == DBNOPROC)
{
    /* The registered procedure must not exist */
    fprintf(stderr, "ERROR: price_change \
                doesn't exist!\n");
}
```

See also `dbregwatch`, `dbregwatchlist`, `dbreghandle`, `dbregexec`

## dbregparam

**Description** Define or describe a registered procedure parameter.

**Syntax** `RETCODE dbregparam(dbproc,param_name, type, datalen, data)`

```
DBPROCESS  *dbproc;
char        *param_name;
int         type;
DBINT       datalen;
BYTE        *data;
```

**Parameters**

**dbproc**  
A pointer to the `DBPROCESS` structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and the server.

**param\_name**  
A pointer to the parameter name.

When creating a registered procedure, *param\_name* is required.

When executing a registered procedure, *param\_name* may be `NULL`. In this case, the registered procedure will expect to receive its parameters in the order in which they were originally defined.

**type**  
A symbolic value indicating the datatype of the parameter. Legal data types are: `SYBINT1`, `SYBINT2`, `SYBINT4`, `SYBREAL`, `SYBFLT8`, `SYBCHAR`, `SYBBINARY`, `SYBVARCHAR`, `SYBDATETIME4`, `SYBDATETIME`, `SYBMONEY4`, and `SYBMONEY`.

Note that `SYBTEXT` and `SYBIMAGE` are not legal datatypes for parameters.

**datalen**

The length of the parameter.

When creating a registered procedure:

- *datalen* can be used to indicate that no default value is being supplied for this parameter. To indicate no default, pass *datalen* as DBNODEFAULT.
- *datalen* can be used to indicate that the default value for a parameter is NULL. This is different from having no default. To indicate a NULL default, pass *datalen* as 0.

When executing a registered procedure:

- *datalen* may be 0. In this case, *data* is ignored and NULL is passed to the registered procedure for this parameter.

**data**

A pointer to the parameter.

When creating a registered procedure, *data* can be used to provide a default value for the parameter. Pass *data* as pointing to the default value. If no default value is desired, pass *datalen* as DBNODEFAULT.

When executing a registered procedure, *data* may be passed as NULL.

**Return value**

SUCCEED or FAIL.

**Usage**

- dbregparam defines a registered procedure parameter. Because a notification procedure is simply a special type of registered procedure, dbregparam also defines a notification procedure parameter.
- dbregparam is called to define registered procedure parameters when a registered procedure is created and to describe the parameters when a registered procedure is executed.

---

**Note** DB-Library/C applications can create only a special type of registered procedure, known as a notification procedure. A notification procedure differs from a normal Open Server registered procedure in that it contains no executable statements. See the dbnpdefine and dbnpcreate reference pages.

---

- Either dbnpdefine, which initiates the process of creating a notification procedure, or dbreginit, which initiates the process of executing a registered procedure, must be called before an application calls dbregparam.

- When creating a registered procedure:
  - To indicate that no default value is being supplied, pass *datalen* as *DBNODEFAULT*. *data* is ignored in this case.
  - To supply a default value of NULL, pass *datalen* as 0. *data* is ignored in this case.
  - To supply a default value that is not NULL pass *datalen* as the length of the value (or -1 if it is a fixed-length type), and *data* as pointing to the value.
- When executing a registered procedure:
  - To pass NULL as the value of the parameter, pass *datalen* as 0. In this case, *data* is ignored.
  - To pass a value for this parameter, pass *datalen* as the length of the value (or -1 if it is a fixed-length type), and *data* as pointing to the value.
- To create a notification procedure, a DB-Library/C application must:
  - Define the procedure using *dbnpdefine*
  - Describe the procedure's parameters, if any, using *dbregparam*
  - Create the procedure using *dbnpcreate*
- This is an example of creating a notification procedure:

```
DBPROCESS    *dbproc;
DBINT         status;
/*
** Let's create a notification procedure called
** "message" which has two parameters:
**      msg      varchar(255)
**      userid   int
**/
/*

** Define the name of the notification procedure
** "message"
**/
dbnpdefine (dbproc, "message", DBNULLTERM);
/* The notification procedure has two parameters:
**      msg      varchar(255)
**      userid   int
** So, define these parameters. Note that both
** of these parameters are defined with a default
```

```
    ** value of NULL. Passing datalen as 0
    ** accomplishes this.
    */

dbregparam (dbproc, "msg", SYBVARCHAR, 0, NULL);
dbregparam (dbproc, "userid", SYBINT4, 0, NULL);

/* Create the notification procedure: */
status = dbnpcreate (dbproc);
if (status == FAIL)
{
    fprintf(stderr, "ERROR: Failed to create \
        message!\n");
}
else
{
    fprintf(stdout, "Success in creating \
        message!\n");
}
```

- To execute a registered procedure, a DB-Library/C application must:
  - Initiate the call using dbreginit
  - Pass the procedure's parameters, if any, using dbregparam
  - Execute the procedure through dbregexec
- This is an example of executing a registered procedure:

```
DBPROCESS    *dbproc;
DBINT         newprice = 55;
DBINT         status;

/*
** Initiate execution of the registered procedure
** "price_change".
**/
dbreginit (dbproc, "price_change", DBNULLTERM);

/*
** The registered procedure has two parameters:
**     name          varchar(255)
**     newprice      int
** So pass these parameters to the registered
** procedure.
**/
dbregparam (dbproc, "name", SYBVARCHAR, 6,
    "sybase");
dbregparam (dbproc, "newprice", SYBINT4, -1,
```



```

        &newprice);

/* Execute the registered procedure: */
status = dbregexec (dbproc, DBNOTIFYALL);
if (status == FAIL)
{
    fprintf(stderr, "ERROR: Failed to execute \
        price_change!\n");
}
else if (status == DBNOPROC)
{
    fprintf(stderr, "ERROR: Price_change does \
        not exist!\n");
}
else
{
    fprintf(stdout, "Success in executing \
        price_change!\n");
}

```

See also `dbreginit`, `dbregexec`, `dbnpdefine`, `dbnpcreate`, `dbregwatch`

## dbregwatch

**Description** Request to be notified when a registered procedure executes.

**Syntax** `RETCODE dbregwatch(dbproc, procedure_name, namelen, options)`

```

DBPROCESS    *dbproc;
DBCHAR       *procedure_name;
DBSMALLINT    namelen;
DBUSMALLINT   options;

```

**Parameters** `dbproc`

A pointer to the `DBPROCESS` structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and the server.

`procedure_name`

A pointer to the name of a registered procedure. The registered procedure must be defined in Open Server.

**namelen**

The length of *procedure\_name*, in bytes. If *procedure\_name* is null-terminated, pass *namelen* as DBNULLTERM.

**options**

A two-byte bitmask: DBWAIT, DBNOWAITONE, or DBNOWAITALL.

If options is passed as DBWAIT, dbregwatch will not return until the DBPROCESS connection reads a synchronous notification that the registered procedure has executed.

If options is passed as DBNOWAITONE, dbregwatch returns -immediately. The DBPROCESS connection will receive an asynchronous notification when the registered procedure executes. The connection will receive only a single notification, even if the registered procedure executes multiple times.

If options is passed as DBNOWAITALL, dbregwatch returns immediately. The DBPROCESS connection will receive an asynchronous notification when the registered procedure executes. The connection will continue to receive notifications, one for each execution of the registered procedure, until it informs Open Server that it no longer wishes to receive them.

**Return value**

SUCCEED, DBNOPROC, or FAIL.

dbregwatch returns FAIL if no handler is installed for the registered procedure.

**Usage**

- dbregwatch informs Open Server that a DBPROCESS connection should be notified when a particular registered procedure executes. Because a notification procedure is simply a special type of registered procedure, dbregwatch also informs Open Server that a DBPROCESS connection should be notified when a particular notification procedure executes.
- The connection can request to be notified synchronously or asynchronously:
  - To request synchronous notification, an application passes *options* as DBWAIT in its call to dbregwatch. In this case, dbregwatch will not return until the DBPROCESS connection reads the notification that the registered procedure has executed.

Open Server will send only a single notification as the result of a synchronous notification request. If the registered procedure executes a second time, after the synchronous request has been satisfied, the client will not receive a second notification, unless another notification request is made.

- To request asynchronous notification, an application passes *options* as DBNOWAITONE or DBNOWAITALL in its call to dbregwatch. In this case, dbregwatch returns immediately. A return code of SUCCEED indicates that Open Server has accepted the request.

If *options* is DBNOWAITONE, Open Server will send only a single notification, even if the registered procedure executes multiple times.

If *options* is DBNOWAITALL, Open Server will continue to send a notification every time the registered procedure executes, until it is informed, using dbregnowatch, that the client no longer wishes to receive them.
- A DBPROCESS connection may be idle, sending commands, reading results, or idle with results pending when an asynchronous registered procedure notification arrives.
  - If the DBPROCESS connection is idle, it is necessary for the application to call dbpoll to allow the connection to read the notification. If a handler for the notification has been installed, it will be called before dbpoll returns.
  - If the DBPROCESS connection is sending commands, the notification is read and the notification handler called during dbsqlxexec or dbsqllok. After the notification handler returns, flow of control continues normally.
  - If the DBPROCESS connection is reading results, the notification is read and the notification handler called either in dbresults or dbnextrow. After the notification handler returns, flow of control continues normally.
  - If the DBPROCESS connection is idle with results pending, the notification is not read until all results in the stream up to the notification have been read and processed by the connection.
- An application must install a handler to process the registered procedure notification before calling dbregwatch. If no handler is installed, dbregwatch returns FAIL. An application can install a notification handler using dbreghandle.

If the handler is uninstalled after the application calls dbregwatch but before the registered procedure notification is received, DB-Library/C raises an error when the notification is received.

- If the procedure referenced by *procedure\_name* is not defined in Open Server, dbregwatch returns DBNOPROC. An application can obtain a list of procedures currently registered in Open Server using dbreglist.
- An application can obtain a list of registered procedures it is watching for using dbregwatchlist.
- This is an example of making a synchronous notification request:

```
DBPROCESS    *dbproc;
DBINT         handlerfunc;
DBINT         ret;

/*
** The registered procedure is defined in Open
** Server as:
**      shutdown    msg_param    varchar(255)
**/

/*
** First install the handler for this registered
** procedure:
**/
dbreghandle(dbproc, "shutdown", DBNULLTERM,
            handlerfunc);

/* Make the notification request and wait: */
ret = dbregwatch(dbproc, "shutdown", DBNULLTERM,
                DBWAIT);

if (ret == FAIL)
{
    fprintf (stderr, "ERROR: dbregwatch() \
                failed!\n");
}
else if (ret == DBNOPROC)
{
    fprintf (stderr, "ERROR: procedure shutdown \
                not defined.\n");
}
else
{
    /*
    ** The registered procedure notification has
    ** been returned, and our registered
    ** procedure handler has already been called.
    **/
}
```

- This is an example of making an asynchronous notification request:

```

DBPROCESS    *dbproc;
DBINT         handlerfunc;
DBINT         ret;

/*
** The registered procedure is defined in Open
** Server as:
**      shutdown    msg_param    varchar(255)
**/

/*
** First install the handler for this registered
** procedure:
**/
dbreghandle(dbproc, "shutdown", DBNULLTERM,
            handlerfunc);

/* Make the asynchronous notification request: */
ret = dbregwatch(dbproc, "shutdown", DBNULLTERM,
                DBNOWAITALL);

if (ret == FAIL)
{
    fprintf (stderr, "ERROR:  dbregwatch() \
                failed!\n");
}
else if (ret == DBNOPROC)
{
    fprintf (stderr, "ERROR: procedure shutdown \
                not defined.\n");
}

/*
** Since we are making use of the asynchronous
** registered procedure notification mechanism,
** the application can continue doing other work
** while waiting for the notification.  All we
** have to do is call dbpoll() once in a while to
** read the registered procedure notification
** when it arrives.
**/

while (1)
{
    /* Have dbpoll() block for one second */
    dbpoll (NULL, 1000, NULL, &ret);

    /*

```

```
        ** If we got the notification, then exit
        ** the loop
        */
        if (ret == DBNOTIFICATION)
            break;
        /* Handle other program tasks here */
    }
```

See also `dbpoll`, `dbregexec`, `dbregparam`, `dbreglist`, `dbregwatchlist`, `dbregnowatch`

## dbregwatchlist

|              |   |
|--------------|---|
| Description  | Return a list of registered procedures that a DBPROCESS is watching for.  |
| Syntax       | <code>RETCODE dbregwatchlist(dbproc)</code>   |
| Parameters   | <p><code>DBPROCESS *dbproc;</code></p> <p><code>dbproc</code></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>  |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"><li>• <code>dbregwatchlist</code> returns a list of registered procedures that a DBPROCESS connection is watching for. Because a notification procedure is simply a special type of registered procedure, the list returned by <code>dbregwatchlist</code> will include notification procedures.</li><li>• The list of registered procedures is returned as rows that an application must explicitly process after calling <code>dbregwatchlist</code>. Each row represents the name of a single registered procedure for which the DBPROCESS has requested notification. A row contains a single column of type SYBVARCHAR.</li><li>• The following code fragment illustrates how <code>dbregwatchlist</code> might be used in an application:</li></ul> |

```
DBPROCESS      *dbproc;
DBCHAR          *procedurename;
DBINT           ret;

/* Request the list of procedures */
if( (ret = dbregwatchlist(dbproc)) == FAIL)
```

```

{
    /* Handle failure here */
}
dbresults(dbproc);
while( dbnextrow(dbproc) != NO_MORE_ROWS )
{
    procedurename = (DBCHAR *)dbdata(dbproc, 1);
    procedurename[dbdatlen(dbproc, 1)] = '\0';

    fprintf(stdout, "we're waiting for \
        procedure '%s'.\n", procedurename);
}
/* All done */

```

See also `dbregwatch`, `dbresults`, `dbnextrow`

## dbresults

|              |  |
|--------------|--|
| Description  | Set up the results of the next query.  |
| Syntax       | <pre>RETCODE dbresults(dbproc)</pre> <pre>DBPROCESS *dbproc;</pre>   |
| Parameters   | <p><code>dbproc</code></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>  |
| Return value | <p>SUCCEED, FAIL or NO_MORE_RESULTS.</p> <p><code>dbresults</code> returns NO_MORE_RESULTS if all commands in the buffer have already been processed. The most common reason for failing is a runtime error, such as a database permission violation.</p> <p>The number of commands in the command buffer determines whether <code>dbsqlxexec</code> or <code>dbresults</code> traps a runtime error. If the buffer contains only a single command, a runtime error will cause <code>dbsqlxexec</code> to fail. If the command buffer contains multiple commands, a runtime error will not cause <code>dbsqlxexec</code> to fail. Instead, the <code>dbresults</code> call that processes the command causing the runtime error fails.</p> |

The situation is a bit more complicated for runtime errors and stored procedures. A runtime error on an execute command may cause dbresults to fail, in accordance with the rule given in the previous paragraph. A runtime error on a statement inside a stored procedure will not cause dbresults to fail, however. For example, if the stored procedure contains an insert statement and the user does not have insert permission on the database table, the insert statement fails, but dbresults will still return SUCCEED. To check for runtime errors inside stored procedures, use the dbretstatus routine to look at the procedure's return status, and trap relevant server messages inside your message handler.

#### Usage

- This routine sets up the next command in the command batch for processing. The application program calls it after dbsqlxec or dbsqlqok returns SUCCEED. The first call to dbresults will always return either SUCCEED or NO\_MORE\_RESULTS if the call to dbsqlxec or dbsqlqok has returned SUCCEED. Once dbresults returns SUCCEED, the application typically processes any result rows with dbnextrow.
- If a command batch contains only a single command, and that command does not return rows, for example a “use database” command, a DB-Library/C application does not have to call dbresults to process the results of the command. However, if the command batch contains more than one command, a DB-Library/C application must call dbresults once for every command in the batch, whether or not the command returns rows.

dbresults must also be called at least once for any stored procedure executed in a command batch, whether or not the stored procedure returns rows. If the stored procedure contains more than one Transact-SQL select, then dbresults must be called once for each select.

To ensure that dbresults is called the correct number of times, Sybase strongly recommends that dbresults always be called in a loop that terminates when dbresults returns NO\_MORE\_RESULTS.

---

**Note** All Transact-SQL commands are considered commands by dbresults. For a list of Transact-SQL commands, see the *Adaptive Server Enterprise Reference Manual*.

---

- To cancel the remaining results from the command batch (and eliminate the need to continue calling dbresults until it returns NO\_MORE\_RESULTS), call dbcancel.
- To determine whether a particular command is one that returns rows and needs results processing with dbnextrow, call DBROWS after the dbresults call.



- The typical sequence of calls for using `dbresults` with `dbsqlxexec` is:

```

DBINT          xvariable;
DBCHAR         yvariable[10];
RETCODE        return_code;

/* Read the query into the command buffer */
dbcmd(dbproc, "select x = 100, y = 'hello'");

/* Send the query to Adaptive Server Enterprise */
dbsqlxexec(dbproc);

/*
** Get ready to process the results of the query.
** Note that dbresults is called in a loop even
** though only a single set of results is expected.
** This is simply because it is good programming
** practice to always code dbresults calls in loop.
**/

while ((return_code
=dbresults(dbproc) != NO_MORE_RESULTS)
{
    if ((return_code == SUCCEED)
        & & (DBROWS(dbproc) == SUCCEED))
    {
        /* Bind column data to program variables */
        dbbind(dbproc, 1, INTBIND, (DBINT) 0,
            (BYTE *) &xvariable);
        dbbind(dbproc, 2, STRINGBIND, (DBINT) 0,
            yvariable);

        /* Now process each row */
        while (dbnextrow(dbproc) != NO_MORE_ROWS)
        {
            C-code to print or process row data
        }
    }
}

```

The sample program *example1.c* shows how to use `dbresults` to process a multiquery command batch.

- To manage multiple input data streams efficiently, an application can confirm that unread bytes are available, either in the DB-Library network buffer or in the network itself. The application can either:
  - (For UNIX only) call `DBRBUF` to test whether bytes remain in the network buffer, and call `DBIORDESC` and the UNIX `select` to test whether bytes remain in the network itself, or

- (For all systems) call dbpoll.
- Another use for dbresults is to process the results of a remote procedure call made with dbrpcsend. See the dbrpcsend reference page for details.

See also

dbbind, dbcancel, dbnextrow, dbpoll, DBRBUF, dbretstatus, DBROWS, dbrpcsend, dbsqlexec, dbsqllok

## dbretdata

**Description** Return a pointer to a return parameter value generated by a stored procedure.

**Syntax** BYTE \*dbretdata(dbproc, retnum)

```
DBPROCESS *dbproc;  
int retnum;
```

**Parameters**

**dbproc**

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

**retnum**

The number of the return parameter value of interest. The first return value is 1. Values are returned in the same order as the parameters were originally specified in the stored procedure's create procedure statement. (Note that this is not necessarily the same order as specified in the remote procedure call.) When specifying *retnum*, non-return parameters do not count. For example, if the second parameter in a stored procedure is the only return parameter, its *retnum* is 1, not 2.

**Return value**

A pointer to the specified return value. If *retnum* is out of range, dbretdata returns NULL. To determine whether the data really has a null value (and *retnum* is not merely out of range), check for a return of 0 from dbretlen.

**Usage**

- dbretdata returns a pointer to a return parameter value generated by a stored procedure. It is useful in conjunction with remote procedure calls and execute statements on stored procedures.

- Transact-SQL stored procedures can return values for specified “return parameters.” Changes made to the value of a return parameter inside the stored procedure are then available to the program that called the procedure. This is analogous to the “pass by reference” facility available in some programming languages.

For a parameter to function as a return parameter, it must be declared as such within the stored procedure. The `execute` statement or remote procedure call that calls the stored procedure must also indicate that the parameter should function as a return parameter. In the case of a remote procedure call, it is the `dbrpcparam` routine that specifies whether a parameter is a return parameter.

- When executing a stored procedure, the server returns any parameter values immediately after returning all other results. Therefore, the application can call `dbretdata` only after processing the stored procedure’s results by calling `dbresults`, as well as `dbnextrow` if appropriate. (Note that a stored procedure can generate several sets of results—one for each `select` it contains. Before the application can call `dbretdata` or any other routines that process return parameters, it must call `dbresults` and `dbnextrow` as many times as necessary to process all the results.)
- If a stored procedure is invoked with a remote procedure call, the return parameter values are automatically available to the application. If, on the other hand, the stored procedure is invoked with an `execute` statement, the return parameter values are available only if the command batch containing the `execute` statement uses local variables, not constants, for the return parameters.
- The routine `dbnumrets` indicates how many return parameter values are available. If `dbnumrets` returns less than or equal to 0, no return parameter values are available.
- When a stored procedure is invoked with an RPC command (using `dbrpcinit`, `dbrpcparam`, and `dbrpcsend`), then the return parameter values can be retrieved after all other results have been processed. For an example of this usage, see the sample program *example8.c*.
- When a stored procedure has been executed from a batch of Transact-SQL commands (with `dbsqlxexec` or `dbsqlsend`), then other commands might execute after the stored procedure. This situation makes retrieval of return parameter values a little more complicated.

- If you are sure that the stored procedure command is the only command in the batch, then you can retrieve the return parameter values after the dbresults loop, as shown in the sample program *example8.c*.
- If the batch can contain multiple commands, then the return parameter values should be retrieved inside the dbresults loop, after all rows have been fetched with dbnextrow. The code below shows where the return parameters should be retrieved in this situation.

```
while ( (result_code = dbresults(dbproc)
        != NO_MORE_RESULTS)
{
    if (result_code == SUCCEED)
    {
        ... bind rows here ...
        while ((row_code = dbnextrow(dbproc))
                != NO_MORE_ROWS)
        {
            ... process rows here ...
        }
        /* Now check for a return status */
        if (dbhasretstat(dbproc) == TRUE)
        {
            printf("(return status %d)\n",
                    dbretstatus(dbproc));
        }
        /* Now check for return parameter values */
        if (dbnumrets(dbproc) > 0)
        {
            ... retrieve output parameters here ...
        }
    } /* if result_code */
    else
    {
        printf("Query failed.\n");
    }
} /* while dbresults */
```

- The routines below are used to retrieve return parameter values:
  - dbnumrets returns the total number of return parameter values.
  - dbretlen returns the length of a parameter value.
  - dbretname returns the name of a parameter value.
  - dbrettype returns the datatype of a parameter value.

- `dbconvert` converts the value to another datatype, if necessary.

The code fragment below shows how these routines are used together:

```
char    dataval[512];
char    *dataname;
DBINT   datalen;
int i,   numrets;
numrets = dbnumrets(dbproc);

for (i = 1; i <= numrets; i++)
{
    dataname = dbretname(dbproc, i);
    datalen = dbretlen(dbproc, i);
    if (datalen == 0)
    {
        /* The parameter's value is NULL */
        strcpy(dataval, "NULL");
    }
    else
    {
        /*
         ** Convert to char. dbconvert appends a null
         ** terminator because we pass the last
         ** parameter, destlen, as -1.
         */
        result = dbconvert(dbproc,
                           dbrettype(dbproc, i),
                           dbretdata(dbproc, i), datalen,
                           SYBCHAR, (BYTE *)dataval, -1);
    } /* else */
    /* Now print out the converted value */
    if (dataname == NULL || *dataname == '\0')
        printf("\t%s\n", dataval);
    else
        printf("\t%s: %s\n", dataname, dataval);
}
```

See also

`dbnextrow`, `dbnumrets`, `dbresults`, `dbretlen`, `dbretname`, `dbrettype`, `dbrpcinit`, `dbrpcparam`

## dbretlen

|              |  |
|--------------|--|
| Description  | Determine the length of a return parameter value generated by a stored procedure.  |
| Syntax       | <pre>DBINT dbretlen(dbproc, retnum)</pre><br><pre>DBPROCESS *dbproc;<br/>int retnum;</pre>   |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>retnum</b><br/>The number of the return parameter value of interest. The first return value is 1. Values are returned in the same order as the parameters were originally specified in the stored procedure's create procedure statement. (Note that this is not necessarily the same order as specified in the remote procedure call.) When specifying <i>retnum</i>, non-return parameters do not count. For example, if the second parameter in a stored procedure is the only return parameter, its <i>retnum</i> is 1, not 2.</p>  |
| Return value | The length of the specified return parameter value. If <i>retnum</i> is out of range, dbretlen returns -1. If the return value is null, dbretlen returns 0.  |
| Usage        | <ul style="list-style-type: none"><li>• dbretlen returns the length of a particular return parameter value generated by a stored procedure. It is useful in conjunction with remote procedure calls and execute statements on stored procedures.</li><li>• Transact-SQL stored procedures can return values for specified "return parameters." Changes made to the value of a return parameter inside the stored procedure are then available to the program that called the procedure. This is analogous to the "pass by reference" facility available in some programming languages.</li></ul> <p>For a parameter to function as a return parameter, it must be declared as such within the stored procedure. The execute statement or remote procedure call that calls the stored procedure must also indicate that the parameter should function as a return parameter. In the case of a remote procedure call, it is the dbrpcparam routine that specifies whether a parameter is a return parameter.</p> |

- When executing a stored procedure, the server returns any parameter values immediately after returning all other results. Therefore, the application can call `dbretlen` only after processing the stored procedure's results by calling `dbresults`, as well as `dbnextrow` if appropriate. (Note that a stored procedure can generate several sets of results—one for each `select` it contains. Before the application can call `dbretlen` or any other routines that process return parameters, it must call `dbresults` and `dbnextrow` as many times as necessary to process all the results.)
- If the stored procedure is invoked with a remote procedure call, the return parameter values are automatically available to the application. If, on the other hand, the stored procedure is invoked with an `execute` statement, the return parameter values are available only if the command batch containing the `execute` statement uses local variables, not constants, for the return parameters.
- Other routines return additional information about return parameter values:
  - `dbnumrets` returns the total number of return parameter values.
  - `dbretdata` returns a pointer to a parameter value.
  - `dbretname` returns the name of a parameter value.
  - `dbrettype` returns the datatype of a parameter value.
  - `dbconvert` converts the value to another datatype, if necessary.
- For an example of this routine, see the `dbretdata` reference page.

See also

`dbnextrow`, `dbnumrets`, `dbresults`, `dbretdata`, `dbretname`, `dbrettype`, `dbrpcinit`, `dbrpcparam`

## **dbretname**

Description

Determine the name of the stored procedure parameter associated with a particular return parameter value.

Syntax

```
char *dbretname(dbproc, retnum)
```

```
DBPROCESS  *dbproc;  
int         retnum;
```

|              |   |
|--------------|---|
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>retnum</b></p> <p>The number of the return parameter value of interest. The first return value is 1. Values are returned in the same order as the parameters were originally specified in the stored procedure's create procedure statement. (Note that this is not necessarily the same order as specified in the remote procedure call.) When specifying <i>retnum</i>, non-return parameters do not count. For example, if the second parameter in a stored procedure is the only return parameter, its <i>retnum</i> is 1, not 2.</p>   |
| Return value | <p>A pointer to the null-terminated parameter name for the specified return value. If <i>retnum</i> is out of range, <i>dbretname</i> returns NULL.</p>   |
| Usage        | <ul style="list-style-type: none"><li>• <i>dbretname</i> returns a pointer to the null-terminated parameter name associated with a return parameter value from a stored procedure. It is useful in conjunction with remote procedure calls and execute statements on stored procedures.</li><li>• Transact-SQL stored procedures can return values for specified "return parameters." Changes made to the value of a return parameter inside the stored procedure are then available to the program that called the procedure. This is analogous to the "pass by reference" facility available in some programming languages.</li></ul> <p>For a parameter to function as a return parameter, it must be declared as such within the stored procedure. The execute statement or remote procedure call that calls the stored procedure must also indicate that the parameter should function as a return parameter. In the case of a remote procedure call, it is the <i>dbrpcparam</i> routine that specifies whether a parameter is a return parameter.</p> <ul style="list-style-type: none"><li>• When executing a stored procedure, the server returns any parameter values immediately after returning all other results. Therefore, the application can call <i>dbretname</i> only after processing the stored procedure's results by calling <i>dbresults</i>, as well as <i>dbnextrow</i> if appropriate. (Note that a stored procedure can generate several sets of results—one for each select it contains. Before the application can call <i>dbretname</i> or any other routines that process return parameters, it must call <i>dbresults</i> and <i>dbnextrow</i> as many times as necessary to process all the results.)</li></ul> |



- If the stored procedure is invoked with a remote procedure call, the return parameter values are automatically available to the application. If, on the other hand, the stored procedure is invoked with an `execute` statement, the return parameter values are available only if the command batch containing the `execute` statement uses local variables, not constants, for the return parameters.
- Other routines return additional information about return parameter values:
  - `dbnumrets` returns the total number of return parameter values.
  - `dbretdata` returns a pointer to a parameter value.
  - `dbretlen` returns the length of a parameter value.
  - `dbrettype` returns the datatype of a parameter value.
  - `dbconvert` converts the value to another datatype, if necessary.
- For an example of this routine, see the `dbretdata` reference page.

See also

`dbnextrow`, `dbnumrets`, `dbresults`, `dbretdata`, `dbretlen`, `dbrettype`, `dbrpcinit`, `dbrpcparam`

## **dbretstatus**

|              |   |
|--------------|---|
| Description  | Determine the stored procedure status number returned by the current command or remote procedure call.  |
| Syntax       | <pre>DBINT dbretstatus(dbproc)</pre>  |
| Parameters   | <pre>DBPROCESS *dbproc;</pre> <p><code>dbproc</code><br/>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> |
| Return value | The return status number for the current command.   |

Usage

- dbretstatus fetches a stored procedure's status number. All stored procedures that are run on Adaptive Server Enterprise return a status number. Stored procedures that complete normally return a status number of 0. For a list of return status numbers, see the *Adaptive Server Enterprise Reference Manual*.
- The dbhasretstat routine determines whether the current Transact-SQL command or remote procedure call actually generated a return status number. Since status numbers are a feature of stored procedures, only a remote procedure call or a Transact-SQL command that executes a stored procedure can generate a status number.
- When executing a stored procedure, the server returns the status number immediately after returning all other results. Therefore, the application can call dbretstatus only after processing the stored procedure's results by calling dbresults, as well as dbnextrow if appropriate. (Note that a stored procedure can generate several sets of results—one for each select it contains. Before the application can call dbretstatus or dbhasretstat, it must call dbresults and dbnextrow as many times as necessary to process all the results.)
- The order in which the application processes the status number and any return parameter values is unimportant.
- When a stored procedure has been executed from a batch of Transact-SQL commands (with dbsqlxexec or dbsqlsend), then other commands might execute after the stored procedure. This situation makes return-status retrieval a little more complicated.
  - If you are sure that the stored procedure command is the only command in the batch, then you can retrieve the return status after the dbresults loop, as shown in the sample program *example8.c*.
  - If the batch can contain multiple commands, then the return status should be retrieved inside the dbresults loop, after all rows have been fetched with dbnextrow. For an example of how return statuses are retrieved in this situation, see the dbhasretstat reference page.
- For an example of this routine, see the dbhasretstat reference page.

See also

dbhasretstat, dbnextrow, dbresults, dbretdata, dbrpcinit, dbrpcparam, dbrpcsend

## dbrettype

|              |   |
|--------------|---|
| Description  | Determine the datatype of a return parameter value generated by a stored procedure.   |
| Syntax       | <pre>int dbrettype(dbproc, retnum)</pre> <pre>DBPROCESS  *dbproc; int         retnum;</pre>   |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>retnum</b><br/>The number of the return parameter value of interest. The first return value is 1. Values are returned in the same order as the parameters were originally specified in the stored procedure's create procedure statement. (Note that this is not necessarily the same order as specified in the remote procedure call.) When specifying <i>retnum</i>, non-return parameters do not count. For example, if the second parameter in a stored procedure is the only return parameter, its <i>retnum</i> is 1, not 2.</p> |
| Return value | <p>A token value for the datatype of the specified return value.</p> <p>In a few cases, the token value returned by this routine may not correspond exactly with the column's server datatype:</p> <ul style="list-style-type: none"> <li>• SYBVARCHAR is returned as SYBCHAR.</li> <li>• SYBVARBINARY is returned as SYBBINARY.</li> <li>• SYBDATETIME is returned as SYBDATETIME.</li> <li>• SYBMONEYN is returned as SYBMONEY.</li> <li>• SYBFLT8 is returned as SYBFLT8.</li> <li>• SYBINTN is returned as SYBINT1, SYBINT2, or SYBINT4, depending on the actual type of the SYBINTN.</li> </ul> <p>If <i>retnum</i> is out of range, -1 is returned.</p>   |
| Usage        | <ul style="list-style-type: none"> <li>• dbrettype returns the datatype of a return parameter value generated by a stored procedure. It is useful in conjunction with remote procedure calls and execute statements on stored procedures.</li> </ul>  |

- Transact-SQL stored procedures can return values for specified “return parameters.” Changes made to the value of a return parameter inside the stored procedure are then available to the program that called the procedure. This is analogous to the “pass by reference” facility available in some programming languages.

For a parameter to function as a return parameter, it must be declared as such within the stored procedure. The `execute` statement or remote procedure call that calls the stored procedure must also indicate that the parameter should function as a return parameter. In the case of a remote procedure call, it is the `dbrpcparam` routine that specifies whether a parameter is a return parameter.

- When executing a stored procedure, the server returns any parameter values immediately after returning all other results. Therefore, the application can call `dbretype` only after processing the stored procedure’s results by calling `dbresults`, as well as `dbnextrow` if appropriate. (Note that a stored procedure can generate several sets of results—one for each `select` it contains. Before the application can call `dbretype` or any other routines that process return parameters, it must call `dbresults` and `dbnextrow` as many times as necessary to process all the results.)
- If the stored procedure is invoked with a remote procedure call, the return parameter values are automatically available to the application. If, on the other hand, the stored procedure is invoked with an `execute` statement, the return parameter values are available only if the command batch containing the `execute` statement uses local variables, not constants, for the return parameters.
- `dbretype` actually returns an integer token value for the datatype (`SYBCHAR`, `SYBFLT8`, and so on). To convert the token value into a readable token string, use `dbprtype`. See the `dbprtype` reference page for a list of all token values and their equivalent token strings.
- For a list of server datatypes, see `Types` on page 412.
- The routines return additional information about return parameter values:
  - `dbnumrets` returns the total number of return parameter values.
  - `dbretdata` returns a pointer to a parameter value.
  - `dbretlen` returns the length of a parameter value.
  - `dbretname` returns the name of a parameter value.
  - `dbconvert` converts the value to another datatype, if necessary.

- For an example of this routine, see the dbretdata reference page.
- See also      dbnextrow, dbnumrets, dbprtype, dbresults, dbretdata, dbretlen, dbretname, dbrpcinit, dbrpcparam

## DBROWS

- Description      Indicate whether the current command actually returned rows.
- Syntax      RETCODE DBROWS(dbproc)
- Parameters      DBPROCESS \*dbproc;  
dbproc  
A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.
- Return value      SUCCEED or FAIL, indicating whether the current command returned rows.
- Usage
  - This macro determines whether the command currently being processed by dbresults returned any rows. The application can call it after dbresults returns SUCCEED.
  - The application must not call DBROWS after dbnextrow. The macro may return the wrong result at that time.
  - The application can use DBROWS to determine whether it needs to call dbnextrow to process result rows. If DBROWS returns FAIL, the application can skip the dbnextrow calls.
  - The DBCMDROW macro determines whether the current command is one that can return rows (that is, a Transact-SQL select statement or an execute on a stored procedure containing a select).
- See also      DBCMDROW, dbnextrow, dbresults, DBROWTYPE

## DBROWTYPE

- Description      Return the type of the current row.

|              |  |
|--------------|--|
| Syntax       | STATUS DBROWTYPE(dbproc)   |
|              | DBPROCESS *dbproc;   |
| Parameters   | dbproc<br>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.  |
| Return value | Three different types of values can be returned: <ul style="list-style-type: none"> <li>• If the current row is a regular row, REG_ROW is returned.</li> <li>• If the current row is a compute row, the <i>computeid</i> of the row is returned. (See the dbaltbind reference page for information on the <i>computeid</i>.)</li> <li>• If no rows have been read, or if the routine failed for any reason, NO_MORE_ROWS is returned.</li> </ul> |
| Usage        | <ul style="list-style-type: none"> <li>• This macro tells you the type (regular or compute) of the current row. Usually you already know this, since dbnextrow also returns the row type.</li> </ul>   |
| See also     | dbnextrow  |

## dbrpcinit

|             |   |
|-------------|---|
| Description | Initialize a remote procedure call.   |
| Syntax      | RETCODE dbrpcinit(dbproc, rpcname, options)   |
|             | DBPROCESS *dbproc;<br>char *rpcname;<br>DBSMALLINT options;   |
| Parameters  | dbproc<br>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server. |
|             | rpcname<br>A pointer to the name of the stored procedure to be invoked.   |

**options**

A 2-byte bitmask of RPC options. So far, the only option available is `DBRPCRECOMPILE`, which causes the stored procedure to be recompiled before it is executed.

**Return value**

**SUCCEED** or **FAIL**.

**Usage**

- An application can call a stored procedure in two ways: by executing a command buffer containing a Transact-SQL `execute` statement or by making a remote procedure call (RPC).
- Remote procedure calls have a few advantages over `execute` statements:
  - An RPC passes the stored procedure's parameters in their native datatypes, in contrast to the `execute` statement, which passes parameters as ASCII characters. Therefore, the RPC method is faster and usually more compact than the `execute` statement, because it does not require either the application program or the server to convert between native datatypes and their ASCII equivalents.
  - It is simpler and faster to accommodate stored procedure return parameters with an RPC, instead of an `execute` statement. With an RPC, the return parameters are automatically available to the application. (Note, however, that a return parameter must be specified as such when it is originally added to the RPC through the `dbrpcparam` routine.) If, on the other hand, a stored procedure is called with an `execute` statement, the return parameter values are available only if the command batch containing the `execute` statement uses local variables, not constants, as the return parameters. This involves additional parsing each time the command batch is executed.
- To make a remote procedure call, first call `dbrpcinit` to specify the stored procedure that is to be invoked. Then call `dbrpcparam` once for each of the stored procedure's parameters. Finally, call `dbrpcsend` to signify the end of the parameter list. This causes the server to begin executing the specified procedure. You can then call `dbsqlok`, `dbresults`, and `dbnextrow` to process the stored procedure's results. (Note that you will need to call `dbresults` multiple times if the stored procedure contains more than one `select` statement.) After all of the stored procedure's results have been processed, you can call the routines that process return parameters and status numbers, such as `dbretdata` and `dbretstatus`.
- If the procedure being executed resides on a server other than the one to which the application is directly connected, commands executed within the procedure cannot be rolled back.

- For an example of a remote procedure call, see the sample program *example8.c*.

See also [dbnextrow](#), [dbresults](#), [dbretdata](#), [dbretstatus](#), [dbrpcparam](#), [dbrpcsend](#), [dbsqlok](#)

## dbrpcparam

|             |   |
|-------------|---|
| Description | Add a parameter to a remote procedure call. |
|-------------|---|

Syntax RETCODE dbbrpcparam(dbproc, paramname, status, type, maxlen, datalen, value)

```
DBPROCESS *dbproc;
char *paramname;
BYTE status;
int type;
DBINT maxlen;
DBINT datalen;
BYTE *value;
```

|            |   |
|------------|---|
| Parameters | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p> |
|------------|---|

**paramname**  
A pointer to the name of the parameter to be invoked. This name must begin with the “@” character, which prefixes all stored procedure parameter names. As in the Transact-SQL execute statement, the name is optional. If it is not used, it should be specified as NULL. In that case, the order of the `dbprcparam` calls determines the parameter to which each refers.

status

A 1-byte bitmask of RPC-parameter options. So far, the only option available is `DBRPCRETURN`, which signifies that the application program would like this parameter used as a return parameter.



**type**

A symbolic constant indicating the datatype of the parameter (for example, SYBINT1, SYBCHAR, and so on). Parameter values should be sent to the server in a datatype that matches the Adaptive Server Enterprise datatype with which the corresponding stored procedure parameter was defined—see Types on page 412 for a list of type constants and the corresponding Adaptive Server Enterprise datatypes.

**maxlen**

For return parameters, this is the maximum desired byte length for the RPC parameter value returned from the stored procedure. *maxlen* is relevant only for values whose datatypes are not fixed in length—that is, char, text, binary, and image values. If this parameter does not apply (that is, if the *type* is a fixed length datatype such as SYBINT2) or if you do not care about restricting the lengths of return parameters, set *maxlen* to -1. *maxlen* should also be set to -1 for parameters not designated as return parameters.

**datalen**

The length, in bytes, of the RPC parameter to pass to the stored procedure. This length should not count any null terminator.

If *type* is SYBCHAR, SYBVARCHAR, SYBBINARY, SYBVARBINARY, SYBBOUNDARY, or SYBSENSITIVITY, *datalen* must be specified.

Passing *datalen* as -1 for any of these datatypes results in the DBPROCESS referenced by *dbproc* being marked as “dead,” or unusable.

If *type* is a fixed length datatype, for example, SYBINT2, pass *datalen* as -1.

If the value of the RPC parameter is NULL, pass *datalen* as 0, even if *type* is a fixed-length datatype.

**value**

A pointer to the RPC parameter itself. If *datalen* is 0, this pointer will be ignored and treated as NULL. Note that DB-Library does not copy *\*value* into its internal buffer space until the application calls `dbrpcsend`. An application must not write over *\*value* until after it has called `dbrpcsend`.

The value of *type* indicates the datatype of *\*value*. See Types on page 412. For types that have no C equivalent, such as SYBDATETIME, SYBMONEY, SYBNUMERIC, or SYBDECIMAL, use `dbconvert_ps` to initialize *\*value*.

---

**Note** An application must not write over *\*value* until after it has called `dbrpcsend` to send the remote procedure call to the server. This is a functional change from previous versions of DB-Library.

---

|              |  |
|--------------|--|
| Return value | SUCCEED or FAIL.   |
| Usage        | <ul style="list-style-type: none"> <li>• An application can call a stored procedure in two ways: by executing a command buffer containing a Transact-SQL execute statement or by making a remote procedure call (RPC). See the reference page for <i>dbrpcinit</i> for a discussion of the differences between these techniques.</li> <li>• To make a remote procedure call, first call <i>dbrpcinit</i> to specify the stored procedure that is to be invoked. Then call <i>dbrpcparam</i> once for each of the stored procedure's parameters. Finally, call <i>dbrpcsend</i> to signify the end of the parameter list. This causes the server to begin executing the specified procedure. You can then call <i>dbsqlok</i>, <i>dbresults</i>, and <i>dbnextrow</i> to process the stored procedure's results. (Note that you will need to call <i>dbresults</i> multiple times if the stored procedure contains more than one select statement.) After all of the stored procedure's results have been processed, you can call the routines that process return parameters and status numbers, such as <i>dbretdata</i> and <i>dbretstatus</i>.</li> <li>• If <i>type</i> is SYBCHAR, SYBVARCHAR, SYBBINARY, SYBVARBINARY, SYBBOUNDARY, and SYBSENSITIVITY, <i>datalen</i> must be specified. Passing <i>datalen</i> as -1 for any of these datatypes results in the DBPROCESS referenced by <i>dbproc</i> being marked as "dead," or unusable.</li> <li>• If <i>type</i> is SYBNUMERIC or SYBDECIMAL, use <i>dbconvert_ps</i> to initialize the DBNUMERIC or DBDECIMAL value in <i>*value</i> and specify its precision and scale.</li> <li>• If the procedure being executed resides on a server other than the one to which the application is directly connected, commands executed within the procedure cannot be rolled back.</li> <li>• For an example of a remote procedure call, see the sample program <i>example8.c</i>.</li> </ul> |
| See also     | <i>dbnextrow</i> , <i>dbresults</i> , <i>dbretdata</i> , <i>dbretstatus</i> , <i>dbrpcinit</i> , <i>dbrpcsend</i> , <i>dbsqlok</i>   |

## dbrpcsend

|             |   |
|-------------|---|
| Description | Signal the end of a remote procedure call.              |
| Syntax      | <pre>RETCODE dbrpcsend(dbproc) DBPROCESS *dbproc;</pre> |

|              |  |
|--------------|--|
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.</p>  |
| Return value | SUCCEED or FAIL.   |
| Usage        | <ul style="list-style-type: none"><li>• An application can call a stored procedure in two ways: by executing a command buffer containing a Transact-SQL execute statement or by making a remote procedure call (RPC). See the reference page for <code>dbrpcinit</code> for a discussion of the differences between these techniques.</li><li>• To make a remote procedure call, first call <code>dbrpcinit</code> to specify the stored procedure that is to be invoked. Then call <code>dbrpcparam</code> once for each of the stored procedure's parameters. Finally, call <code>dbrpcsend</code> to signify the end of the parameter list. This causes the server to begin executing the specified procedure. You can then call <code>dbsqlok</code>, <code>dbresults</code>, and <code>dbnextrow</code> to process the stored procedure's results. (Note that you will need to call <code>dbresults</code> multiple times if the stored procedure contains more than one select statement.) After all of the stored procedure's results have been processed you can call the routines that process return parameters and status numbers, such as <code>dbretdata</code> and <code>dbretstatus</code>.</li><li>• If the procedure being executed resides on a server other than the one to which the application is directly connected, commands executed within the procedure cannot be rolled back.</li><li>• For an example of a remote procedure call, see the sample program <i>example8.c</i>.</li></ul> |
| See also     | <code>dbnextrow</code> , <code>dbresults</code> , <code>dbretdata</code> , <code>dbretstatus</code> , <code>dbrpcinit</code> , <code>dbrpcparam</code> , <code>dbsqlok</code>  |

## dbrpwclr

|             |  |
|-------------|--|
| Description | Clear all remote passwords from the LOGINREC structure.  |
| Syntax      | <pre>void dbrpwclr(loginrec)</pre><br><pre>LOGINREC    *loginrec;</pre>  |
| Parameters  | <p><b>loginrec</b></p> <p>A pointer to a LOGINREC structure. This pointer will serve as an argument to <code>dbopen</code>. You can allocate a LOGINREC structure by calling <code>dblogin</code>.</p> |

|              |   |
|--------------|---|
| Return value | None.   |
| Usage        | <ul style="list-style-type: none"> <li>A Transact-SQL command batch or stored procedure running on one server may call a stored procedure located on another server. To accomplish this server-to-server communication, the first server, connected to the application through <code>dbopen</code>, actually logs into the second, remote server.</li> </ul> <p><code>dbrpwset</code> allows the application to specify the password to be used when the first server attempts to call the stored procedure on the remote server. Multiple passwords may be specified, one for each server that the first server might need to log in to.</p> <ul style="list-style-type: none"> <li>A single <code>LOGINREC</code> can be used repeatedly, in successive <code>dbopen</code> calls to different servers. <code>dbrpwclr</code> allows the application to remove any remote password information currently in the <code>LOGINREC</code>, so that successive calls to <code>dbopen</code> can contain different remote password information (specified with <code>dbrpwset</code>).</li> </ul> |
| See also     | <code>dblogin</code> , <code>dbopen</code> , <code>dbrpwset</code> , <code>DBSETLAPP</code> , <code>DBSETLHOST</code> , <code>DBSETLPWD</code> , <code>DBSETLUSER</code>  |

## dbrpwset

|             |   |
|-------------|---|
| Description | Add a remote password to the <code>LOGINREC</code> structure.   |
| Syntax      | <pre>RETCODE dbrpwset(loginrec, srvname, password, pwlen)</pre> <p> <code>LOGINREC</code>    <code>*loginrec;</code><br/> <code>char</code>        <code>*srvname;</code><br/> <code>char</code>        <code>*password;</code><br/> <code>int</code>          <code>pwlen;</code> </p>   |
| Parameters  | <p><code>loginrec</code></p> <p>A pointer to a <code>LOGINREC</code> structure. This pointer will serve as an argument to <code>dbopen</code>. You can allocate a <code>LOGINREC</code> structure by calling <code>dblogin</code>.</p> <p><code>srvname</code></p> <p>The name of a server. A server's name is stored in the <code>srvname</code> column of its <code>sys.servers</code> system table. When the first server calls a stored procedure located on the server designated by <code>srvname</code>, it will use the specified password to log in. If <code>srvname</code> is <code>NULL</code>, the specified password will be considered a "universal" password, to be used with any server that does not have a password explicitly specified for it.</p> |

|              |  |
|--------------|--|
|              | password   |
|              | The password that the first server will use to log in to the specified server.   |
|              | pwlen  |
|              | The length of the password in bytes.   |
| Return value | SUCCEED or FAIL.   |
|              | This routine may fail if the addition of the specified password would overflow the LOGINREC's remote password buffer. (The remote password buffer is 255 bytes long. Each password's entry in the buffer consists of the password itself, the associated server name, and 2 extra bytes.)  |
| Usage        | <ul style="list-style-type: none"> <li>A Transact-SQL command batch or stored procedure running on one server may call a stored procedure located on another server. To accomplish this server-to-server communication, the first server, connected to the application through dbopen, actually logs into the second, remote server and performs a remote procedure call.</li> </ul> <p>dbrpwset allows the application to specify the password to be used when the first server attempts to call the stored procedure on the remote server. Multiple passwords may be specified, one for each server that the first server might need to log in to.</p> <ul style="list-style-type: none"> <li>If the application has not specified a remote password for a particular server the password defaults to the one set with DBSETLPWD (or a null value, if DBSETLPWD has not been called). This behavior may be fine if the application's user has the same password on multiple servers.</li> <li>dbrpwclr clears all remote passwords from the LOGINREC.</li> </ul> |
| See also     | dblogin, dbopen, dbrpwclr, DBSETLAPP, DBSETLHOST, DBSETLPWD, DBSETLUSER  |

## dbsafestr

|             |  |
|-------------|--|
| Description | Double the quotes in a character string.   |
| Syntax      | <pre>RETCODE dbsafestr(dbproc, src, srclen, dest, destlen,                   quotetype)</pre> <pre>DBPROCESS      *dbproc; char            *src; DBINT          srclen; char            *dest;</pre> |

|            |           |   |
|------------|-----------|---|
|            | DBINT     | destlen;  |
|            | int       | quotetype;  |
| Parameters | dbproc    | A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server. |
|            | src       | A pointer to the original string.   |
|            | srclen    | The length of <i>src</i> , in bytes. If <i>srclen</i> is -1, <i>src</i> is assumed to be null-terminated.   |
|            | dest      | A pointer to a programmer-supplied buffer to contain the resulting string. <i>dest</i> must be large enough for the resulting string plus a null terminator.  |
|            | destlen   | The length of the programmer-supplied buffer to contain the resulting string. If <i>destlen</i> is -1, <i>dest</i> is assumed to be large enough to hold the resulting string.  |
|            | quotetype | The type of quotes to double. Table 2-24 lists the possible values for <i>quotetype</i> .   |

**Table 2-24: Values for quotetype**

| Value of quotetype | dbsafestr  |
|--------------------|--|
| DBSINGLE           | Doubles all single quotes (') in <i>src</i>        |
| DBDOUBLE           | Doubles all double quotes (") in <i>src</i>        |
| DBBOTH             | Doubles all single and double quotes in <i>src</i> |

|              |   |
|--------------|---|
| Return value | SUCCEED or FAIL.<br><br>dbsafestr fails if the resulting string is too large for <i>dest</i> , or if an invalid <i>quotetype</i> is specified.  |
| Usage        | <ul style="list-style-type: none"><li>dbsafestr doubles the single and/or double quotes found in a character string. This is useful when specifying literal quotes within a character string.</li></ul> |
| See also     | dbcmd, dbfcmd   |

## dbsechandle

**Description** Install user functions to handle secure logins.

**Syntax** RETCODE \*dbsechandle(type, handler)

```
DBINT      type;
INTFUNCPTR (*handler());
```

**Parameters** *type*  
An integer variable with one of the symbolic values shown in Table 2-25.

**Table 2-25: Values for type (dbsechandle)**

| Value of type | dbsechandle   |
|---------------|---|
| DBENCRYPT     | Installs a function to handle password encryption   |
| DBLABELS      | Installs a function to handle login security labels |

**handler**

A pointer to the user function that DB-Library will call whenever the corresponding type of secure login needs to be handled.

If *handler* is NULL and *type* is DBENCRYPT, DB-Library will use its default encryption handler.

If *handler* is NULL and *type* is DBLABELS, dbsechandle uninstalls any current label handler.

**Return value** SUCCEED or FAIL.

**Usage**

- dbsechandle installs user functions to handle secure logins.
- An application can use dbsechandle to install functions to handle two types of secure logins:
  - Encrypted password secure logins  
In this type of secure login, the server provides the client with a key. The client uses the key to encrypt a password, which it then returns to the server.
  - Security label secure logins  
In this type of secure login, the server asks the client for identifying security labels, which the client then provides.

**Encrypted password secure logins**

- If *type* is DBENCRYPT, dbsechandle installs the function that DB-Library will call when encrypting user passwords.

- DB-Library will perform password encryption only if DBSETLENCRYPT has been called prior to calling dbopen.
- DB-Library will call its default encryption handler if a user function has not been installed.
- Typically, a user function does not need to be installed for password encryption. This is because DB-Library's default encryption handler allows an application to perform password encryption when connecting to an Adaptive Server Enterprise.
- A user-defined encryption handler should be installed by applications that are gateways. The encryption handler will be responsible for taking the encryption key returned by the remote server, passing it back to the client, reading the encrypted password from the client, and returning the encrypted password to DB-Library so that DB-Library can pass it on to the remote server.
- An encryption handler should be declared as shown in the example below. Encryption handlers on the Windows platform must be declared with CS\_PUBLIC. For portability, callback handlers on other platforms should be declared CS\_PUBLIC as well. Here is a sample declaration:

```
RETCODE CS_PUBLIC encryption_handler(dbproc, pwd,
                                     pwrlen, enc_key, keylen, outbuf, buflen, outlen)
DBPROCESS      *dbproc;
BYTE           *pwd;
DBINT          pwrlen;
BYTE           *enc_key;
DBINT          keylen;
BYTE           *outbuf;
DBINT          buflen;
DBINT          *outlen;
```

where:

- *dbproc* is the DBPROCESS.
- *pwd* is the user password to be encrypted.
- *pwrlen* is the length of the user's password.
- *enc\_key* is the key to be used during encryption.
- *keylen* is the length of the encryption key.
- *outbuf* is a buffer in which the callback can place the encrypted password. This buffer will be allocated and freed by DB-Library.
- *buflen* is the length of the output buffer.



- *outlen* is a pointer to a DBINT. The encryption handler should set *\*outlen* to the length of the encrypted password.
- An encryption handler should return SUCCEED to indicate that the password was encrypted successfully. If the encryption handler returns a value other than SUCCEED, DB-Library will abort the connection attempt.

#### Security label secure logins

- If type is DBLABELS, dbsechandle installs a function that DB-Library will call to get login security labels.
- DB-Library will send login security labels only if DBSETLABELLED has been called prior to calling dbopen.
- There are two ways for an application to define security labels:
  - The application can call dbsetsecurity one time for each label it wants to define. Most applications will use this method.
  - The application can call dbsechandle to install a user-supplied function to generate security labels. Typically, only gateway applications will use this method.

If an application uses both methods, the labels defined through dbsetsecurity and the labels generated by the user-supplied function are sent to the server at the same time.

- DB-Library calls an application's label handler during the connection process, in response to a server request for login security labels. Each time it is called, the label handler returns a single label. DB-Library sends these labels, together with any labels previously defined using dbsetsecurity, to the server.
- DB-Library does not have a default label handler.
- A user-defined label handler should be installed by applications that are gateways. The label handler will be responsible for reading the client's login security labels and passing them on to DB-Library so that DB-Library can pass them on to the remote server.
- A label handler should be declared as shown in the example below. Label handlers on the Windows platform must be declared with CS\_PUBLIC. For portability, callback handlers on other platforms should be declared CS\_PUBLIC as well. Here is a sample declaration:

```
RETCODE CS_PUBLIC label_handler(dbproc, namebuf,  
                                nbufen, valuebuf, vbufen, namelen, valuelen)
```

```
DBPROCESS    *dbproc;  
DBCHAR       *namebuf;  
DBINT        nbuflen;  
DBCHAR       *valuebuf;  
DBINT        vbuflen;  
DBINT        *namelen;  
DBINT        *valuelen;
```

where:

- *dbproc* is the DBPROCESS.
- *namebuf* is a buffer in which the handler can place the name of the login security label. This buffer is allocated and freed by DB-Library.
- *nbuflen* is the length of the *namebuf* buffer.
- *valuebuf* is a buffer in which the handler can place the value of the login security label. This buffer is allocated and freed by DB-Library.
- *vbuflen* is the length of the *valuebuf* buffer.
- *namelen* is a pointer to a DBINT. The label handler should set *\*namelen* to the length of the label name placed in *namebuf*.
- *valuelen* is a pointer to a DBINT. The label handler should set *\*valuelen* to the length of the label value placed in *valuebuf*.
- Table 2-26 lists the return values that are legal for a security label handler. A security label handler must return one of these values.

**Table 2-26: Return values for security label handlers**

| Label handler<br>return value | Indicates   |
|-------------------------------|---|
| DBMORELABEL                   | The label handler has set the name and value of a login security label.<br><br>DB-Library should call the label handler again to get an additional label. |
| DBENDLABEL                    | The label handler has set the name and value of a login security label.<br><br>DB-Library should not call the label handler again.                        |
| DBERRLABEL                    | A label handler error has occurred. DB-Library should abort the connection attempt.   |

See also

DBSETLENCRYPT, dbopen.

## dbsendpassthru

|              |  |
|--------------|--|
| Description  | Send a TDS packet to a server.   |
| Syntax       | <pre>RETCODE dbsendpassthru(dbproc, send_bufp)  DBPROCESS  *dbproc; DBVOIDPTR  send_bufp;</pre>  |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and the server.</p> <p><b>send_bufp</b><br/>A pointer to a buffer containing the TDS packet to be sent to the server. A packet has a default size of 512 bytes. This size may be changed using DBSETLPACKET.</p>  |
| Return value | DB_PASSTHRU_MORE, DB_PASSTHRU_EOM, or FAIL.  |
| Usage        | <ul style="list-style-type: none"> <li>• dbsendpassthru sends a TDS (Tabular Data Stream) packet to a server.</li> <li>• TDS is an application protocol used for the transfer of requests and request results between clients and servers. Under ordinary circumstances, a DB-Library/C application does not have to deal directly with TDS, because DB-Library/C manages the data stream.</li> <li>• dbrecvpassthru and dbsendpassthru are useful in gateway applications. When an application serves as the intermediary between two servers, it can use these routines to pass the TDS stream from one server to the other, eliminating the process of interpreting the information and re-encoding it.</li> <li>• dbsendpassthru sends a packet of bytes from the buffer to which <i>send_bufp</i> points. Most commonly, <i>send_bufp</i> will be <i>*recv_bufp</i> as returned by dbrecvpassthru. <i>send_bufp</i> may also be the address of a user-allocated buffer containing the packet to be sent.</li> <li>• A packet has a default size of 512 bytes. An application can change its packet size using DBSETLPACKET. See the dbgetpacket and DBSETLPACKET reference pages.</li> <li>• dbsendpassthru returns DB_PASSTHRU_EOM if the TDS packet in the buffer is marked as EOM (End Of Message). If the TDS packet is not the last in the stream, dbsendpassthru returns DB_PASSTHRU_MORE.</li> </ul> |

- A DBPROCESS connection that is used for a dbsendpassthru operation cannot be used for any other DB-Library/C function until DB\_PASSTHRU\_EOM is received.
- This is a code fragment using dbsendpassthru:

```
/*
** The following code fragment illustrates the
** use of dbsendpassthru() in an Open Server
** gateway application. It will continually get
** packets from a client, and pass them through
** to the remote server.
**
** The routine srv_rcvpssthru() is the Open
** Server counterpart required to complete this
** passthru operation.
*/

DBPROCESS      *dbproc;
SRV_PROC       *srvproc;
int             ret;
BYTE           *packet;
while(1)
{
    ret = srv_rcvpssthru(srvproc, &packet,
        (int *)NULL);

    if( ret == SRV_S_PASSTHRU_FAIL )
    {
        fprintf(stderr, "ERROR - \
            srv_rcvpssthru failed in \
            lang_execute.\n");
        exit();
    }
    /*
    ** Now send the packet to the remote server
    */
    if( dbsendpassthru(dbproc, packet) == FAIL )
    {
        fprintf(stderr, "ERROR - dbsendpassthru\
            failed in lang_execute.\n");
        exit();
    }
    /*
    ** We've sent the packet, so let's see if
    ** there's any more.
    */
    if( ret == SRV_S_PASSTHRU_MORE )
```

```
        continue;
    else
        break;
}
```

See also `dbrecvpass thru`

## **dbservcharset**

|              |   |
|--------------|---|
| Description  | Get the name of the server character set.   |
| Syntax       | <code>char *dbservcharset(dbproc)</code><br><br><code>DBPROCESS *dbproc;</code>   |
| Parameters   | <code>dbproc</code><br>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library/C uses to manage communications and data between the front end and the server.   |
| Return value | A pointer to the null-terminated name of the server's character set, or <code>NULL</code> in case of error.   |
| Usage        | <ul style="list-style-type: none"><li>• <code>dbservcharset</code> returns the name of the server's character set.</li><li>• DB-Library/C clients can use a different character set than the server or servers to which they are connected. If a client and server are using different character sets, and the server supports character translation for the client's character set, it will perform all conversions to and from its own character set when communicating with the client.</li><li>• An application can inform the server what character set it is using <code>DBSETLCHARSET</code>.</li><li>• To determine if the server is performing character set translations, an application can call <code>dbcharsetconv</code>.</li><li>• To get the name of the client character set, an application can call <code>dbgetcharset</code>.</li></ul> |
| See also     | <code>dbcharsetconv</code> , <code>dbgetcharset</code> , <code>DBSETLCHARSET</code>   |

## dbsetavail

|              |  |
|--------------|--|
| Description  | Marks a DBPROCESS as being available for general use.  |
| Syntax       | <pre>void dbsetavail(dbproc)</pre><br><pre>DBPROCESS  *dbproc;</pre>   |
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>  |
| Return value | None.  |
| Usage        | This routine marks the DBPROCESS as being available for general use. Any subsequent calls to DBISAVAIL will return “TRUE”, until some use is made of the DBPROCESS. Many DB-Library routines automatically set the DBPROCESS to “not available.” This is useful when many different parts of a program are attempting to share a single DBPROCESS. |
| See also     | DBISAVAIL  |

## dbsetbusy

|             |  |
|-------------|--|
| Description | Call a user-supplied function when DB-Library is reading from the server.  |
| Syntax      | <pre>void dbsetbusy(dbproc, busyfunc)</pre><br><pre>DBPROCESS  *dbproc; int         *(*busyfunc)();</pre>  |
| Parameters  | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>busyfunc</b></p> <p>The user-supplied function that DB-Library will call whenever it accesses the server. DB-Library calls <i>busyfunc()</i> with a single parameter—a pointer to the DBPROCESS from the dbsetbusy call.</p> <p><i>busyfunc()</i> returns a pointer to a function that returns an integer.</p> |

Return value           None.

- Usage
- This routine associates a user-supplied function with the specified *dbproc*. The user-supplied function will be automatically called whenever DB-Library is reading or waiting to read output from the server. For example, an application may want to print a message whenever the server is accessed. *dbsetbusy* will cause the user-supplied function *busyfunc()* to be called in this case.
  - Similarly, *dbsetidle* may also be used to associate a user-supplied function, *idlefunc()*, with a *dbproc*. *idlefunc()* will be automatically called whenever DB-Library has finished reading output from the server.
  - The server sends result data to the application in packets of 512 bytes. (The final packet in a set of results may be less than 512 bytes.) DB-Library calls *busyfunc()* at the beginning of each packet and *idlefunc()* at the end of each packet. If the output from the server spans multiple packets, *busyfunc()* and *idlefunc()* will be called multiple times.
  - Here is an example of defining and installing *busyfunc()* and *idlefunc()*:

---

**Note** The application functions *busyfunc()* and *idlefunc()* are callback event handlers and must be declared as *CS\_PUBLIC* for the Windows platform. For portability, callback handlers on other platforms should be declared *CS\_PUBLIC* as well.

---

```

/*
** busyfunc returns a pointer to a function that
** returns an integer.
*/
int      (*busyfunc())();
void      idlefunc();

int      counterfunc();
...

main()
{
    DBPROCESS      *dbproc;
    ...

    dbproc = dbopen(login, NULL);

    /*
    ** Now that we have a DBPROCESS, install the
    ** busy-function and the idle-function.
    */
    dbsetbusy(dbproc, busyfunc);

```

```
        dbsetidle(dbproc, idlefunc);

        dbcmd(dbproc, "select * from sysdatabases");
        dbcmd(dbproc, " select * from sysobjects");
        dbsqlxec(dbproc);

        /*
         ** DB-Library calls busyfunc() for the first time
         ** during dbsqlxec(). Depending on the size of the
         ** results, it may call busyfunc() again during
         ** processing of the results.
         */

        while (dbresults(dbproc) != NO_MORE_RESULTS)
            dbprrow(dbproc);

        /*
         ** DB-Library calls idlefunc() each time a packet
         ** of results has been received. Depending on the
         ** size of the results, it may call idlefunc()
         ** multiple times during processing of the results.
         */
        ...
    }

int CS_PUBLIC (*busyfunc(dbproc)) ()
    DBPROCESS dbproc;
{
    printf("Waiting for data...\n");

    return(counterfunc);
}

void CS_PUBLIC idlefunc(procptr, dbproc)

/*
 ** idlefunc's first parameter is a pointer to a
 ** routine that returns an integer. This is the same
 ** pointer that busyfunc returns.
 */
    int (*procptr) ();
    DBPROCESS *dbproc;
{
    int count;

    printf("Data is ready.\n");
    count = (*procptr) ();

    printf ("Counterfunc has been called %d %s.\n",
        count, (count == 1 ? "time" : "times"));
}
```



```

    }

    int counterfunc()
    {
        static      int      counter = 0;

        return(++counter);
    }

```

See also `dbsetidle`

## dbsetconnect

**Description** Specify server connection information to use instead of directory services.

**Syntax** `RETCODE dbsetconnect(service_type, net_type, net_name, machine_name, port)`

```

char      *service_type;
char      *net_type;
char      *net_name;
char      *machine_name;
char      *port;

```

**Parameters**

**service\_type**  
The type of connection. Default values are:

- “master” specifies a master line, which is used by server applications to listen for client queries.
- “query” specifies a query line, which is used by client applications to find servers.

**net\_type**  
The name of the network protocol. Valid values are:

- “tcp” for TCP/IP – all UNIX platforms
- “decnet” for DECnet

**net\_name**  
Descriptor of the network. Open Client and Open Server do not currently use `net_name`; it is a placeholder should Sybase need to define this information in the future. For TCP/IP networks, the `net_name` is set to “ether.”

**machine\_name**

The network name of the node, or machine, that the server is running on. The maximum number of characters for *machine\_name* depends on the protocol specified in the entry:

- For TCP/IP, the maximum is 32.
- For DECnet, the maximum is 6.

Use the `/bin/hostname` command on UNIX platforms to determine the network name of the machine you are logged in to.

**port**

Port used by the server to receive queries. The TCP/IP and DECnet protocols specify this element differently:

- TCP/IP: Registered port numbers range from 1024 to 49151. Sybase recommends to use a port number from this range.
- DECnet: Valid object numbers range from 128 to 253. Object names are also valid.

Use the `netstat` command to check which port numbers are in use.

**Return value**

SUCCEED or FAIL.

**Usage**

- This routine lets the application specify connection information such as service type, network protocol type, network name of the server, server name, and port number required to connect to the server. This connection information is used for every subsequent call to `dbopen`.
- If `dbsetconnect` is used, DSQUERY and normal directory services lookup for a server entry is bypassed.
- If `dbsetconnect` has not been called, the connection information is found using directory services. The default directory service is the *interfaces* file for UNIX and the *sql.ini* file for Windows. Other directory services may be specified using the configuration file, *libtcl.cfg*.
- See the *Open Client and Open Server Configuration Guide*.

**See also**

`dbopen`

## dbsetdefcharset

**Description**

Set the default character set for an application.

|              |  |
|--------------|--|
| Syntax       | <code>RETCODE dbsetdefcharset(charset)</code><br><br><code>char      *charset;</code>  |
| Parameters   | <code>charset</code><br>The name of the character set to use. <i>charset</i> must be a null-terminated character string.   |
| Return value | SUCCEED or FAIL.   |
| Usage        | <ul style="list-style-type: none"><li>• <code>dbsetdefcharset</code> sets an application's default character set.</li><li>• DB-Library uses a default character set when no DBPROCESS structure is available or when localization information for a DBPROCESS structure's character set cannot be found.</li><li>• If an application does not call <code>dbsetdefcharset</code>, its default character set is the character set of the first DBPROCESS connection opened, or <code>iso_1</code> if no DBPROCESS is open.</li><li>• If an application plans to call both <code>dbsetdefcharset</code> and <code>dbsetdeflang</code>, it must call <code>dbsetdefcharset</code> first.</li></ul> |
| See also     | <code>dbsetdeflang</code> , <code>dbsetdefcharset</code> , <code>dblogin</code> , <code>dbopen</code>  |

## dbsetdeflang

|              |   |
|--------------|---|
| Description  | Set the default language name for an application.   |
| Syntax       | <code>RETCODE dbsetdeflang(language)</code><br><br><code>char      *language;</code>  |
| Parameters   | <code>language</code><br>The name of the national language to use. <i>language</i> must be a null-terminated character string.  |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"><li>• <code>dbsetdeflang</code> sets an application's default national language.</li><li>• DB-Library uses a default language when no DBPROCESS structure is available or when localization information for a DBPROCESS structure's language cannot be found.</li></ul> |

- If an application does not call `dbsetdeflang`, its default language is the language of the first DBPROCESS connection opened, or `us_english` if no DBPROCESS is open.

See also

DBSETLNATLANG

## dbsetidle

Description

Call a user-supplied function when DB-Library is finished reading from the server.

Syntax

```
void dbsetidle(dbproc, idlefunc)
```

```
DBPROCESS  *dbproc;  
void        (*idlefunc)();
```

Parameters

`dbproc`

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

`idlefunc`

The user-supplied function that will be called by DB-Library whenever the server has finished sending data to the host. DB-Library calls `idlefunc()` with two parameters—the return value from `busyfunc()` (a pointer to a function that returns an integer) and a pointer to the DBPROCESS from the `dbsetidle` call.

*idlefunc()* returns void.

Return value

None.

Usage

- This routine associates a user-supplied function with the specified *dbproc*. The user-supplied function will be automatically called when DB-Library is finished reading or waiting to read a packet of output from the server. For example, an application may want to print a message whenever the server has finished sending data to the host. `dbsetidle` will cause the user-supplied function `idlefunc()` to be called in this case.
- Similarly, `dbsetbusy` may also be used to associate a user-supplied function, `busyfunc()`, with a *dbproc*. `busyfunc()` will be automatically called whenever DB-Library is reading or waiting to read a packet of output from the server.

- The server sends result data to the application in packets of 512 bytes. (The final packet in a set of results may be less than 512 bytes.) DB-Library calls *busyfunc()* at the beginning of each packet and *idlefunc()* at the end of each packet. If the output from the server spans multiple packets, *busyfunc()* and *idlefunc()* will be called multiple times.
- See the *dbsetbusy* reference page for an example of defining and installing *busyfunc()* and *idlefunc()*.

See also

*dbsetbusy*

## dbsetifile

|              |   |
|--------------|---|
| Description  | Specify the name and location of the Sybase interfaces file.  |
| Syntax       | <pre>void dbsetifile(filename)</pre>  |
| Parameters   | <pre>char    *filename;</pre> <p>filename</p> <p>The name of the interfaces file that gets searched during every subsequent call to <i>dbopen</i>. If this parameter is NULL, DB-Library will revert to the default file name.</p>  |
| Return value | None.   |
| Usage        | <ul style="list-style-type: none"><li>• This routine lets the application specify the name and location of the interfaces file that will be searched during every subsequent call to <i>dbopen</i>. The interfaces file contains the name and network address of every server available on the network.</li><li>• If <i>dbsetifile</i> has not been called, a call to <i>dbopen</i> initiates the following default behavior: DB-Library attempts to use a file named <i>interfaces</i> in the directory named by the SYBASE environment variable or logical name. If SYBASE has not been set, DB-Library attempts to use a file called <i>interfaces</i> in the home directory of the user named “sybase.”</li><li>• See the <i>Open Client and Open Server Configuration Guide</i>.</li></ul> |

---

**Note** On non-UNIX platforms, client applications may use a method to find server address information that is different from the UNIX *interfaces* file. See the *Open Client and Open Server Configuration Guide* for detailed information on how clients connect to servers.

---

See also                      dbopen

**dbsetinterrupt**

Description                      Calls user-supplied functions to handle interrupts while waiting on a read from the server.

Syntax                              void dbsetinterrupt(dbproc, chkintr, hndlintr)

```
DBPROCESS  *dbproc;
int          (*chkintr)();
int          (*hndlintr)();
```

Parameters                      dbproc  
                                 A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

chkintr  
                                 A pointer to the user function that DB-Library calls to check whether an interrupt is pending. DB-Library calls it periodically while waiting on a read from the server. DB-Library calls chkintr() with a single parameter—a pointer to the DBPROCESS from the dbsetinterrupt call.  
  
chkintr() must return “TRUE” or “FALSE”.

hndlintr  
                                 A pointer to the user function that DB-Library calls if an interrupt is returned. DB-Library calls hndlintr() with a single parameter—a pointer to the DBPROCESS from the dbsetinterrupt call.

Table 2-27 lists the legal return values of hndlintr:

**Table 2-27: Return values for the hndlintr() function**

| Return value | To indicate   |
|--------------|---|
| INT_EXIT     | Abort the program. (Note to UNIX programmers: DB-Library will not leave a core file.    |
| INT_CANCEL   | Abort the current command batch. Results are not flushed from the DBPROCESS connection. |
| INT_CONTINUE | Continue to wait for the server response.   |

Return value                      None.

**Usage**

- DB-Library does non-blocking reads from the server. While waiting for a read from the server, it calls the `chkintr()` function to see if an interrupt is pending. If `chkintr()` returns “TRUE” and a handler has been installed as the `hndlintr()` for `dbsetinterrupt`, `hndlintr()` is called. `dbsetinterrupt` is provided so that the programmer can substitute alternative interrupt handling for the time that the host program is waiting on reads from the server.
- Depending on the return value from `hndlintr()`, DB-Library performs one the following actions:
  - Sends an attention to the server, causing the server to discontinue processing (`INT_CANCEL`). For details, see “Canceling from the interrupt handler” on page 315.
  - Continues reading from the server (`INT_CONTINUE`).
  - Exits the program (`INT_EXIT`).

**Canceling from the interrupt handler**

- If `hndlintr()` returns `INT_CANCEL`, DB-Library sends an attention token to the server. This causes the server to discontinue command processing. The server may send additional results that have already been computed. When control returns to the mainline code, the mainline code should do one of the following:
  - Flush the results using `dbcancel`
  - Process the results normally
- You cannot call `dbcancel` in your interrupt handler, because this will cause output from the server to DB-Library to become out of sync. The steps below describe a correct method to cancel from the interrupt handler.
  - Associate an *int\_canceled* flag with the `DBPROCESS` structure. Use `dbsetuserdata` to install a pointer to the flag in the `DBPROCESS`, and `dbgetuserdata` to get the address of the flag.
  - Code `hndlintr()` to set the *int\_canceled* flag to indicate whether or not it is returning `INT_CANCEL`.
  - In the mainline code, check the flag before each call to `dbresults` or `dbnextrow`. When the *int\_canceled* flag indicates that `hndlintr()` has aborted the server command, the mainline code should call `dbcancel` and clear the flag.

## Example

- Here are example *chkintr()* and *hndlintr()* routines:

---

**Note** The applications *chkintr()* and *hndlintr()* routines are callback functions and must be declared as `CS_PUBLIC` for the Windows platform. For portability, callback handlers on other platforms should be declared `CS_PUBLIC` as well.

---

```
int CS_PUBLIC  chkintr(dbproc)
DBPROCESS    *dbproc;
{
    /*
     ** This routine assumes that the application
     ** sets the global variable
     ** "OS_interrupt_happened" upon catching
     ** an interrupt using some operating system
     ** facility.
     */
    if (OS_interrupt_happened)
    {
        /*
         ** Clear the interrupt flag, for
         ** future use.
         */
        OS_interrupt_happened = FALSE;
        return(TRUE);
    }
    else
        return(FALSE);
}

int CS_PUBLIC  hndlintr(dbproc)
DBPROCESS    *dbproc;
{
    char    response[10];
    DBBOOL *int_canceled;
    /*
     ** We assume that a DBBOOL flag has been
     ** attached to dbproc with dbsetuserdata.
     */
    int_canceled = (DBBOOL *) dbgetuserdata(dbproc);
```



```

if (int_canceled == (DBBOOL *)NULL)
{
    printf("Fatal Error: no int_cancel flag \
        in the DBPROCESS\n");
    return(INT_EXIT);
}
*int_canceled = FALSE;
printf("\nAn interrupt has occurred. Do you \
    want to:\n\n");
printf("\t1) Abort the program\n");
printf("\t2) Cancel the current query\n");
printf("\t3) Continue processing the current\
    query's results\n\n");
printf("Press 1, 2, or 3, followed by the \
    return key: ");
gets(response);
switch(response[0])
{
    case '1':
        return(INT_EXIT);
        break;
    case '2':
        *int_canceled = TRUE;
        return(INT_CANCEL);
        break;
    case '3':
        return(INT_CONTINUE);
        break;
    default:
        printf("Response not understood. \
            Aborting program.\n");
        return(INT_EXIT);
        break;
}
}

```

See also `dbcancel`, `dbgetuserdata`, `dbsetuserdata`, `dbsetbusy`, `dbsetidle`

## DBSETLAPP

Description Set the application name in the LOGINREC structure.

Syntax RETCODE DBSETLAPP(loginrec, application)

|              |  |
|--------------|--|
|              | LOGINREC    *loginrec;<br>char        *application;  |
| Parameters   | <p>loginrec<br/>A pointer to a LOGINREC structure, which will be passed as an argument to dbopen. You can allocate a LOGINREC structure by calling dblogin.</p> <p>application<br/>The application name that will be sent to the server. It must be a null-terminated character string. The maximum length of the string, not including the null terminator, is 30 characters.</p>   |
| Return value | SUCCEED or FAIL.   |
| Usage        | <ul style="list-style-type: none"><li>• This macro sets the application field in the LOGINREC structure. For it to have any effect, it must be called before dbopen.</li><li>• It is not necessary to call this routine. By default, the application name will be a null value.</li><li>• The server uses the application name in its sysprocesses table to help identify your process. If you set the application name, you will see it if you query the sysprocesses table in the master database.</li></ul> |
| See also     | dblogin, dbopen, DBSETLHOST, DBSETLPWD, DBSETLUSER   |

## DBSETLCHARSET

|             |  |
|-------------|--|
| Description | Set the character set in the LOGINREC structure.   |
| Syntax      | RETCODE DBSETLCHARSET(loginrec, char_set)  |
|             | LOGINREC    *loginrec;<br>DBCHAR      *char_set;   |
| Parameters  | <p>loginrec<br/>A pointer to a LOGINREC structure to be passed as an argument to dbopen. LOGINREC structures are obtained using dblogin.</p> |

**char\_set**

The name of the character set the client will use. *char\_set* must be a null-terminated string. Default values for *char\_set* include “iso\_1” for ISO-8859-1 (most platforms), “cp850” for Code Page 850 (IBM RS/6000), and “roman8” for the Roman8 character set (HP platforms).

To indicate that no character set conversion is desired, pass *char\_set* as NULL.

**Return value**

SUCCEED or FAIL.

**Usage**

- DBSETLCHARSET sets the client character set in a LOGINREC structure.
- DB-Library/C clients may use a different character set than the server or servers to which they are connected. DBSETLCHARSET is used to inform the server what character set a client is using.
- Because the LOGINREC is passed as a parameter in the dbopen call that establishes the client’s connection with a server, DBSETLCHARSET must be called before dbopen to have any effect.
- The server will perform all conversions to and from its own character set when communicating with a client using a different character set.
- If no conversion is desired, call DBSETLCHARSET with *char\_set* as NULL.

**See also**

dbgetcharset, dblogin, dbopen

## DBSETLENCRYPT

**Description**

Specify whether or not network password encryption is to be used when logging into Adaptive Server Enterprise.

**Syntax**

RETCODE DBSETLENCRYPT(loginrec, enable)

```
LOGINREC    *loginrec;
DBBOOL      enable;
```

**Parameters****loginrec**

A pointer to a LOGINREC structure, which will be passed as an argument to dbopen. You can allocate a LOGINREC structure by calling dblogin.

**enable**

A boolean value (“true” or “false”) specifying whether or not the server should request an encrypted password at login time.

|              |   |
|--------------|---|
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"><li>• DBSETLENCRYPT specifies whether or not network password encryption is to be used when logging into Adaptive Server Enterprise. If an application does not call DBSETLENCRYPT, password encryption is not used.</li><li>• Network password encryption provides a protected mechanism for authenticating a user's identity.</li><li>• If an application specifies that network password encryption is to be used, then when the application attempts to open a connection:<ul style="list-style-type: none"><li>• No password is sent with the initial connection request. At this time, the client indicates to the server that encryption is desired.</li><li>• The server replies to the connection request with an encryption key.</li><li>• DB-Library uses the key to encrypt the user's password and remote passwords, if any, and sends the encrypted passwords back to the server.</li><li>• The server uses the key to decrypt the encrypted passwords and either accepts or rejects the login attempt.</li></ul></li><li>• If password encryption is not specified, then when an application attempts to open a connection:<ul style="list-style-type: none"><li>• A password is included with the connection request.</li><li>• The server either accepts or rejects the login attempt.</li></ul></li></ul> |
| See also     | dbsechandle   |

## DBSETLHOST

|             |   |
|-------------|---|
| Description | Set the host name in the LOGINREC structure.  |
| Syntax      | RETCODE DBSETLHOST(loginrec, hostname)  |
|             | LOGINREC    *loginrec;<br>char        *hostname;  |
| Parameters  | loginrec<br>A pointer to a LOGINREC structure, which will be passed as an argument to dbopen. You can allocate a LOGINREC structure by calling dblogin. |

|                     |   |
|---------------------|---|
|                     | <b>hostname</b><br>The host name that will be sent to the server. It must be a null-terminated character string. The maximum length of the string, not including the null terminator, is 30 characters.   |
| <b>Return value</b> | SUCCEED or FAIL.  |
| <b>Usage</b>        | <ul style="list-style-type: none"><li>• This macro sets the host name in the LOGINREC structure. For it to have any effect, it must be called before dbopen.</li><li>• The host name will show up in the sysprocesses table in the master database.</li><li>• It is not necessary to call this routine. If it is not called, DB-Library will set the default value for the host name. This default value will generally be a version of the host machine's name provided by the operating system.</li></ul> |
| <b>See also</b>     | dblogin, dbopen, DBSETLAPP, DBSETLPWD, DBSETLUSER   |

## DBSETLMUTUALAUTH

|                     |   |
|---------------------|---|
| <b>Description</b>  | Enables or disables mutual authentication of the connection's security mechanism.   |
| <b>Syntax</b>       | RETCODE DBSETLMUTUALAUTH(loginrec, enable)<br><br>LOGINREC           *loginrec;<br>DBBOOL            *enable;   |
| <b>Parameters</b>   | <b>loginrec</b><br>A pointer to a LOGINREC structure, which is passed as an argument to dbopen. You can allocate a LOGINREC structure by calling dblogin.<br><br><b>enable</b><br>A boolean value ("true" or "false") specifying whether or not the server should enable mutual authentication. |
| <b>Return value</b> | SUCCEED or FAIL.  |
| <b>Usage</b>        | <ul style="list-style-type: none"><li>• For DBSETLMUTUALAUTH to take effect, it must be called before dbopen() and DBSETLNETWORKAUTH must be enabled.</li><li>• If DBSETLMUTUALAUTH is not called, mutual authentication is disabled by default.</li></ul>                                      |
| <b>See also</b>     | dblogin, DBSETLNETWORKAUTH, DBSETLSERVERPRINCIPAL   |

## DBSETLNATLANG

|              |   |
|--------------|---|
| Description  | Set the national language name in the LOGINREC structure.   |
| Syntax       | RETCODE DBSETLNATLANG(loginrec, language)   |
|              | <pre>LOGINREC    *loginrec;<br/>char        *language;</pre>  |
| Parameters   | <p>loginrec<br/>A pointer to a LOGINREC structure to be passed as an argument to dbopen. LOGINREC structures are obtained using dblogin.</p> <p>language<br/>The name of the national language to use. <i>language</i> must be a null-terminated character string.</p>                              |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"><li>• This macro sets the user language in the LOGINREC structure. If you wish to set a particular user language, call DBSETLNATLANG before dbopen.</li><li>• Call DBSETLNATLANG only if you do not wish to use the server's default national language.</li></ul> |
| See also     | dblogin, dbopen, dbsetdeflang   |

## DBSETLNETWORKAUTH

|              |   |
|--------------|---|
| Description  | Enables or disables network-based authentication.   |
| Syntax       | RETCODE DBSETLNETWORKAUTH(loginrec, enable)   |
|              | <pre>LOGINREC    *loginrec;<br/>DBBOOL      *enable;</pre>  |
| Parameters   | <p>loginrec<br/>A pointer to a LOGINREC structure, is passed as an argument to dbopen. You can allocate a LOGINREC structure by calling dblogin.</p> <p>enable<br/>A boolean value (“true” or “false”) specifying whether or not the server should enable network authentication.</p> |
| Return value | SUCCEED or FAIL.  |

|          |  |
|----------|--|
| Usage    | If DBSETLNETWORKAUTH is not called, network authentication is disabled by default. |
| See also | dblogin, DBSETLMUTUALAUTH, DBSETLSERVERPRINCIPAL                                   |

## dbsetloginfo

|              |   |
|--------------|---|
| Description  | Transfer TDS login information from a DBLOGININFO structure to a LOGINREC structure.  |
| Syntax       | <pre>RETCODE dbsetloginfo(loginrec, loginfo)</pre><br><pre>LOGINREC    *login;<br/>DBLOGININFO *loginfo;</pre>  |
| Parameters   | <p><b>login</b><br/>A pointer to a LOGINREC structure. This pointer will be passed as an argument to dbopen. You can allocate a LOGINREC structure by calling dblogin.</p> <p><b>loginfo</b><br/>A pointer to a DBLOGININFO structure that contains login parameter information.</p>  |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"><li>• dbsetloginfo transfers TDS login information from a DBLOGININFO structure to a LOGINREC structure. After the information is transferred, dbsetloginfo frees the DBLOGININFO structure.</li><li>• An application needs to call dbsetloginfo only if (1) it is an Open Server gateway application and (2) it is using TDS passthrough.</li><li>• TDS (Tabular Data Stream) is an application protocol used for the transfer of requests and request results between clients and servers.</li><li>• When a client connects directly to a server, the two programs negotiate the TDS format they will use to send and receive data. When a gateway application uses TDS passthrough, the application forwards TDS packets between the client and a remote server without examining or processing them. For this reason, the remote server and the client must agree on a TDS format to use.</li></ul> |

- dbsetloginfo is the second of four calls, two of them Server Library calls, that allow a client and remote server to negotiate a TDS format. The calls, which can only be made in a SRV\_CONNECT event handler, are described here:
  - srv\_getloginfo allocates a DBLOGININFO structure and fills it with TDS information from a client SRV\_PROC.
  - dbsetloginfo transfers the TDS information retrieved by srv\_getloginfo from the DBLOGININFO structure to a DB-Library/C LOGINREC structure, and then frees the DBLOGININFO structure. After the information is transferred, the application can use this LOGINREC structure in the dbopen call that establishes its connection with the remote server.
  - dbgetloginfo transfers the remote server's response to the client's TDS information from a DBPROCESS structure into a newly-allocated DBLOGININFO structure.
  - srv\_setloginfo sends the remote server's response, retrieved by dbgetloginfo, to the client, and then frees the DBLOGININFO structure.
- This is an example of a SRV\_CONNECT handler preparing a remote connection for TDS passthrough:

```
RETCODE      connect_handler(srvproc)
SRVPROC      *srvproc;
{
    SYBLOGINFO  *loginfo;
    LOGINREC    *loginrec;
    DBPROCESS   *dbproc;

    /*
     ** Get the TDS login information from the
     ** client SRV_PROC.
     */
    srv_getloginfo(srvproc, &loginfo);

    /* Get a LOGINREC structure */
    loginrec = dblogin();

    /*
     ** Initialize the LOGINREC with the logininfo
     ** from the SRV_PROC.
     */
    dbsetloginfo(loginrec, loginfo);

    /* Connect to the remote server */
    dbproc = dbopen(loginrec, REMOTE_SERVER_NAME)
```



```

/*
** Get the TDS login response information from
** the remote connection.
*/
dbgetlogininfo(dbproc, &logininfo);

/*
** Return the login response information to
** the SRV_PROC.
*/
srv_setlogininfo(srvproc, logininfo);

/* Accept the connection and return */
srv_senddone(srvproc, 0, 0, 0);
return(SRV_CONTINUE);
}

```

See also `dbgetlogininfo`, `dbrecvpass thru`, `dbsendpass thru`

## dbsetlogintime

|              |   |
|--------------|---|
| Description  | Set the number of seconds that DB-Library waits for a server response to a request for a DBPROCESS connection.  |
| Syntax       | RETCODE dbsetlogintime(seconds)   |
|              | int          seconds;   |
| Parameters   | seconds<br>The timeout value—the number of seconds that DB-Library waits for a login response before timing out. A timeout value of 0 represents an infinite timeout period.  |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"> <li>• This routine sets the length of time in seconds that DB-Library will wait for a login response after calling <code>dbopen</code>. The default timeout value is 60 seconds.</li> <li>• When a connection attempt is made between a client and a server, there are two ways in which the connection can fail (assuming that the system is correctly configured): <ul style="list-style-type: none"> <li>• The machine that the server is supposed to be on is running correctly and the network is running correctly.</li> </ul> </li> </ul> |

In this case, if there is no server listening on the specified port, the machine the server is supposed to be on will signal the client, through a network error, that the connection cannot be formed. Regardless of `dbsetlogintime`, the connection fails.

- The machine that the server is on is down.

In this case, the machine that the server is supposed to be on will not respond. Because “no response” is not considered to be an error, the network will not signal the client that an error has occurred. However, if `dbsetlogintime` has been called to set a timeout period, a timeout error will occur when the client fails to receive a response within the set period.

See also `dberrhandle`, `dbsettime`

## DBSETLPACKET

**Description** Set the TDS packet size in an application’s LOGINREC structure.

**Syntax** RETCODE DBSETLPACKET(login, packet\_size)

```
LOGINREC    *login;  
short       packet_size;
```

**Parameters** `login`  
A pointer to the LOGINREC structure to be passed as an argument to `dbopen` when logging in to the server. An application can obtain a LOGINREC structure using `dblogin`.

*packet\_size*

The packet size being requested, in bytes. The server will set the actual packet size to a value less than or equal to this requested size.

**Return value** SUCCEED or FAIL.

**Usage**

- DBSETLPACKET sets the packet size field in an application’s LOGINREC structure. When the application logs into the server, the server sets the TDS packet size for that DBPROCESS connection to be equal to or less than the value of this field. The packet size is set to a value less than the value of the packet size field if the server is experiencing space constraints. Otherwise, the packet size will be equal to the value of the field.

- If an application sends or receives large amounts of text or image data, a packet size larger than the default 512 bytes may improve efficiency, since it results in fewer network reads and writes.
- To determine the packet size that the server has set, an application can call `dbgetpacket`.
- TDS (Tabular Data Stream) is an application protocol used for the transfer of requests and request results between clients and servers.
- TDS data is sent in fixed-size chunks, called packets. TDS packets have a default size of 512 bytes. The only way an application can change the TDS packet size is through `DBSETLPACKET`. If `DBSETLPACKET` is not called, all `DBPROCESS` connections in an application will use the default size.
- Different `DBPROCESS` connections in an application may use different packet sizes. To set different packet sizes for `DBPROCESS` connections, an application can either:
  - Change the packet size in a single `LOGINREC` between the `dbopen` calls that create the `DBPROCESS` connections, or
  - Set different packet sizes in multiple `LOGINREC` structures, and use these different `LOGINREC` structures when creating the `DBPROCESS` connections.
- Because the actual packet size for a `DBPROCESS` connection is set when the `DBPROCESS` is created, calls to `DBSETLPACKET` will have no effect on the packet sizes of `DBPROCESS`s already allocated using `dbopen`.

See also `dblogin`, `dbopen`, `dbgetpacket`

## DBSETLPWD

Description Set the user server password in the `LOGINREC` structure.

Syntax `RETCODE DBSETLPWD(loginrec, password)`

```
LOGINREC  *loginrec;  
char      *password;
```

Parameters `loginrec`

A pointer to a `LOGINREC` structure, which will be passed as an argument to `dbopen`. You can allocate a `LOGINREC` structure by calling `dblogin`.

|              |   |
|--------------|---|
|              | <p>password</p> <p>The password that will be sent to the server. It must be a null-terminated character string. The maximum length of the string, not including the null terminator, is 30 characters.</p>  |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"><li>• This macro sets the user server password in the LOGINREC structure. For it to have any effect, it must be called before <code>dbopen</code>.</li><li>• By default, the password field of the LOGINREC has a null value. Therefore, you do not need to call this routine if the password is a null value.</li><li>• DB-Library does not automatically blank out the password in <i>loginrec</i> after a call to <code>dbopen</code>. Therefore, if you want to minimize the risk of having a readable password in your DB-Library program, you should set <i>password</i> to something else after you call <code>dblogin</code>.</li></ul> |
| See also     | <code>dblogin</code> , <code>dbopen</code> , <code>DBSETLAPP</code> , <code>DBSETLHOST</code> , <code>DBSETLUSER</code>   |

## DBSETLSERVERPRINCIPAL

|              |   |          |            |      |        |
|--------------|---|----------|------------|------|--------|
| Description  | Sets the server's principal name in the LOGINREC structure, if required.  |          |            |      |        |
| Syntax       | <code>DBSETLSERVERPRINCIPAL(loginrec, name)</code>  |          |            |      |        |
|              | <table><tr><td>LOGINREC</td><td>*loginrec;</td></tr><tr><td>char</td><td>*name;</td></tr></table>   | LOGINREC | *loginrec; | char | *name; |
| LOGINREC     | *loginrec;  |          |            |      |        |
| char         | *name;  |          |            |      |        |
| Parameters   | <p><code>loginrec</code></p> <p>A pointer to a LOGINREC structure, which is passed as an argument to <code>dbopen</code>. You can allocate a LOGINREC structure by calling <code>dblogin</code>.</p> <p><code>name</code></p> <p>The server's principal name. The maximum length of the string, not including the null terminator, is 255 characters.</p> |          |            |      |        |
| Return value | SUCCEED or FAIL.  |          |            |      |        |
| Usage        | <ul style="list-style-type: none"><li>• For <code>DBSETLSERVERPRINCIPAL</code> to take effect, it must be called before <code>dbopen()</code> and <code>DBSETLNETWORKAUTH</code> must be enabled.</li><li>• If <code>DBSETLSERVERPRINCIPAL</code> is not called, the server name is set as the principal name.</li></ul>                                  |          |            |      |        |
| See also     | <code>dblogin</code> , <code>DBSETLMUTUALAUTH</code> , <code>DBSETLNETWORKAUTH</code>   |          |            |      |        |

## DBSETLUSER

|              |   |
|--------------|---|
| Description  | Set the user name in the LOGINREC structure.  |
| Syntax       | RETCODE DBSETLUSER(loginrec, username)  |
|              | <pre>LOGINREC    *loginrec; char         *username;</pre>   |
| Parameters   | <p>loginrec<br/>A pointer to a LOGINREC structure, which will be passed as an argument to dbopen. You can allocate a LOGINREC structure by calling dblogin.</p> <p>username<br/>The user name that will be sent to the server. It must be a null-terminated character string. The maximum length of the string, not including the null terminator, is 30 characters. The server will use <i>username</i> to determine who is attempting the connection. The server usernames are defined in the syslogins table in the master database.</p> |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"><li>• This macro sets the user name in the LOGINREC structure. For it to have any effect, it must be called before dbopen.</li><li>• In most environments, this macro is optional. If it is not called, DB-Library will generally set the default value for the user name.</li></ul>  |
|              | <hr/> <p><b>Note</b> On <i>UNIX</i>: the user name defaults to the UNIX login name.</p> <p>On <i>MPE/XL</i>: The user name defaults to the value of the system environment variable HPUSER.</p> <hr/>   |
| See also     | dblogin, dbopen, DBSETLHOST, DBSETLPWD, DBSETLAPP   |

## dbsetmaxprocs

|             |   |
|-------------|---|
| Description | Set the maximum number of simultaneously open DBPROCESS structures. |
| Syntax      | RETCODE dbsetmaxprocs(maxprocs)                                     |
|             | <pre>int         maxprocs;</pre>                                    |

|              |  |
|--------------|--|
| Parameters   | <p>maxprocs</p> <p>The new limit on simultaneously open DBPROCESS structures for this particular program.</p>  |
| Return value | SUCCEED or FAIL.   |
| Usage        | <ul style="list-style-type: none"><li>• A DB-Library program has a maximum number of simultaneously open DBPROCESS structures. By default, this number is 25. The program may change this limit by calling dbsetmaxprocs.</li><li>• The program may find out what the current limit is by calling dbgetmaxprocs.</li></ul> |
| See also     | dbgetmaxprocs, dbopen  |

## dbsetnull

|             |  |
|-------------|--|
| Description | Define substitution values to be used when binding null values.  |
| Syntax      | <pre>RETCODE dbsetnull(dbproc, bindtype, bindlen, bindval)</pre> <pre>DBPROCESS  *dbproc;<br/>int         bindtype;<br/>int         bindlen;<br/>BYTE        *bindval;</pre>   |
| Parameters  | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>bindtype</b></p> <p>A symbolic value specifying the type of variable binding to which the substitute value will apply. (See the reference page for dbbind.)</p> <p><b>bindlen</b></p> <p>The length in bytes of the substitute value you are supplying. DB-Library ignores it in all cases except CHARBIND and BINARYBIND. All the other types are either fixed length or have a special terminator or embedded byte-count that provides the length of the data.</p> |

`bindval`

A generic BYTE pointer to the value you want to use as a null substitution value. `dbsetnull` makes a copy of the value, so you can free this pointer anytime after this call.

Return value

SUCCESS or FAIL.

`dbsetnull` returns FAIL if you give it an unknown *bindtype*. It will also fail if the specified DBPROCESS is dead.

Usage

- The `dbbind` and `dbaltbind` routines bind result column values to program variables. After the application calls them, calls to `dbnextrow` and `dbgetrow` automatically copy result values into the variables to which they are bound. If the server returns a null value for one of the result columns, DB-Library automatically places a substitute value into the result variable.
- Each DBPROCESS has a list of substitute values for each of the binding types. Table 2-28 lists the default substitution values:

**Table 2-28: Default null substitution values**

| Binding type      | Null substitution value                            |
|-------------------|--|
| TINYBIND          | 0  |
| SMALLBIND         | 0  |
| INTBIND           | 0  |
| CHARBIND          | Empty string (padded with blanks)                  |
| STRINGBIND        | Empty string (padded with blanks, null-terminated) |
| NTBSTRINGBIND     | Empty string (null-terminated)                     |
| VARYCHARBIND      | Empty string                                       |
| BINARYBIND        | Empty array (padded with zeros)                    |
| VARYBINBIND       | Empty array  |
| DATETIMEBIND      | 8 bytes of zeros                                   |
| SMALLDATETIMEBIND | 8 bytes of zeros                                   |
| MONEYBIND         | \$0.00   |
| SMALLMONEYBIND    | \$0.00   |
| FLT8BIND          | 0.0  |
| REALBIND          | 0.0  |
| DECIMALBIND       | 0.0 (with default scale and precision)             |
| NUMERICBIND       | 0.0 (with default scale and precision)             |
| BOUNDARYBIND      | Empty string (null-terminated)                     |
| SENSITIVITYBIND   | Empty string (null-terminated)                     |

- dbsetnull lets you provide your own null substitution values. When you call dbsetnull to change a particular null substitution value, the new value will remain in force for the specified DBPROCESS until you change it with another call to dbsetnull.
- The dbconvert routine also uses the current null substitution values when it needs to set a destination variable to null.
- The dbnullbind routine allows you to associate an indicator variable with a bound column. DB-Library will set the indicator value to indicate null data values or conversion errors.

See also

dbaltbind, dbbind, dbconvert, dbnullbind, Types on page 412

## dbsetopt

Description

Set a server or DB-Library option.



|            |  |
|------------|--|
| Syntax     | <pre> RETCODE dbsetopt(dbproc, option, char_param,                   int_param)  DBPROCESS *dbproc; int        option; char       *char_param; int        int_param; </pre>  |
| Parameters | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server. If <i>dbproc</i> is NULL, the option will be set for all active DBPROCESS structures.</p> <p><b>option</b><br/>The option that is to be turned on. See Options on page 407 for the list of options.</p> <p><b>char_param</b><br/>Certain options take parameters. For example, the DBOFFSET option takes as its parameter the construct for which offsets are to be returned:</p> <pre>dbsetopt (dbproc, DBOFFSET, "compute", -1)</pre> <p>The DBBUFFER option takes as its parameter the number of rows to be buffered:</p> <pre>dbsetopt (dbproc, DBBUFFER, "500", -1)</pre> <p><i>char_param</i> must always be a character string enclosed in quotes, even in the case of a numeric value, as in the DBBUFFER example. If an invalid parameter is specified for one of the server options, this will be discovered the next time a command buffer is sent to the server. The <code>dbsqlxexec</code> or <code>dbsqlsend</code> call fails, and DB-Library will invoke the user-installed message handler. If an invalid parameter is specified for one of the DB-Library options (DBBUFFER or DBTEXTLIMIT), the <code>dbsetopt</code> call itself fails.</p> <p>If the option takes no parameters, <i>char_param</i> must be NULL.</p> <p><b>int_param</b><br/>Some options require an additional parameter, <i>int_param</i>, which is the length of the character string passed as <i>char_param</i>. Currently, only DBPRCOLSEP, DBPRLINESEP, and DBPRPAD require this parameter.</p> <p>If <i>int_param</i> is not required, pass it as -1.</p> <p><b>Return value</b><br/>SUCCEED or FAIL.</p> |

dbsetopt fails if *char\_param* is invalid for one of the DB-Library options. However, an invalid *char\_param* for a server option will not cause dbsetopt to fail, because such a parameter does not get validated until the command buffer is sent to the server.

- Usage
- This routine sets server and DB-Library options. Although server options may be set and cleared directly through SQL, the application should instead use dbsetopt and dbclopt to set and clear options. This provides a uniform interface for setting both server and DB-Library options. It also allows the application to use the dbisopt function to check the status of an option.
  - dbsetopt does not immediately set the option. The option is set the next time a command buffer is sent to the server (by invoking dbsqlxexec or dbsqlsend).
  - For a list of each option and its default status, see Options on page 407.
- See also dbclopt, dbisopt, Options on page 407

## dbsetrow

Description Set a buffered row to “current.”

Syntax STATUS dbsetrow(dbproc, row)

DBPROCESS \*dbproc;  
DBINT row;

Parameters dbproc

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.

row

An integer representing the row number of the row to make current. Row number 1 is the first row returned from the server. This is not necessarily the first row in the row buffer.

Return value MORE\_ROWS, NO\_MORE\_ROWS, or FAIL.

dbsetrow returns:

- MORE\_ROWS if it found *row* in the row buffer, or

## Usage

- NO\_MORE\_ROWS if it did not find *row* in the row buffer or if row buffering is not enabled, or
- FAIL if the *dbproc* DBPROCESS is dead or not enabled.
- *dbsetrow* sets a buffered row to “current.” After *dbsetrow* is called, the application’s next call to *dbnextrow* will read this row.
- *dbgetrow*, another DB-Library/C routine, also sets a specific row in the row buffer to “current.” However, unlike *dbsetrow*, *dbgetrow* reads the row. Any binding of row data to program variables (as specified with *dbbind* and *dbaltbind*) takes effect.
- *dbsetrow* has no effect unless the DB-Library/C option DBBUFFER is on.
- Row buffering provides a way to keep a specified number of server result rows in program memory. Without row buffering, the result row generated by each new *dbnextrow* call overwrites the contents of the previous result row. Row buffering is therefore useful for programs that need to look at result rows in a non-sequential manner. It does, however, carry a memory and performance penalty because each row in the buffer must be allocated and freed individually. Therefore, use it only if you need to. Specifically, the application should only turn the DBBUFFER option on if it calls *dbgetrow* or *dbsetrow*. Note that row buffering has nothing to do with network buffering and is a completely independent issue.
- When row buffering is *not* enabled, the application processes each row as it reads it from the server by calling *dbnextrow* repeatedly until it returns NO\_MORE\_ROWS. When row buffering *is* enabled, the application can use *dbsetrow* to jump to any row that has already been read from the server with *dbnextrow*. Subsequent calls to *dbnextrow* will cause the application to read successive rows in the buffer, starting with the row specified by the *row* parameter. When *dbnextrow* reaches the last row in the buffer, it reads rows from the server again, if there are any. Once the buffer is full, *dbnextrow* does not read any more rows from the server until some of the rows have been cleared from the buffer with *dbclrbuf*.
- The macro DBFIRSTROW, which returns the number of the first row in the row buffer, is useful in conjunction with *dbsetrow*. Thus, the call:

```
dbsetrow(dbproc, DBFIRSTROW(dbproc))
```

sets the current row so that the next call to *dbnextrow* will read the first row in the buffer.

## See also

*dbclrbuf*, *DBCURROW*, *DBFIRSTROW*, *dbgetrow*, *DBLASTROW*, *dbnextrow*, Options on page 407

## dbsettime

|              |   |
|--------------|---|
| Description  | Set the number of seconds that DB-Library will wait for a server response to a SQL command.   |
| Syntax       | <pre>RETCODE dbsettime(seconds)</pre><br><pre>int      seconds;</pre>   |
| Parameters   | <p>seconds</p> <p>The timeout value—the number of seconds that DB-Library waits for a server response before timing out. A timeout value of 0 represents an infinite timeout period.</p>  |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"><li>• This routine sets the length of time in seconds that DB-Library will wait for a server response during calls to <code>dbsqlxexec</code>, <code>dbsqllok</code>, <code>dbresults</code>, and <code>dbnextrow</code>. The default timeout value is 0, which represents an infinite timeout period.</li><li>• <code>dbsettime</code> can be called at any time during the application—before or after a call to <code>dbopen</code>. It takes effect immediately upon being called.</li><li>• To set a timeout value for calls to <code>dbopen</code>, use <code>dbsetlogintime</code>.</li><li>• Note that, after sending a query to the server, <code>dbsqlxexec</code> waits until a response is received or until the timeout period has elapsed. To minimize the time spent in DB-Library waiting for a response from the server, an application can instead call <code>dbsqlsend</code>, followed by <code>dbsqllok</code>.</li><li>• The program can call <code>DBGETTIME</code> to learn the current timeout value.</li><li>• A timeout generates the DB-Library error “SYBETIME.”</li></ul> |
| See also     | <code>dberrhandle</code> , <code>DBGETTIME</code> , <code>dbsetlogintime</code> , <code>dbsqlxexec</code> , <code>dbsqllok</code> , <code>dbsqlsend</code>  |

## dbsetuserdata

|             |   |
|-------------|---|
| Description | Use a <code>DBPROCESS</code> structure to save a pointer to user-allocated data.                  |
| Syntax      | <pre>void dbsetuserdata(dbproc, ptr)</pre><br><pre>DBPROCESS  *dbproc;<br/>BYTE       *ptr;</pre> |

|              |   |
|--------------|---|
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>ptr</b><br/>A generic BYTE pointer to the user's private data space.</p>   |
| Return value | None.   |
| Usage        | <ul style="list-style-type: none"> <li>• This routine saves, in a DBPROCESS structure, a pointer to user-allocated data. The application can access the data later with the <code>dbgetuserdata</code> routine.</li> <li>• <code>dbsetuserdata</code> allows the application to associate user data with a particular DBPROCESS. This avoids the necessity of using global variables for this purpose. One use for this routine is to handle deadlock, as shown in the example below. This routine is particularly useful when the application has multiple DBPROCESS structures.</li> <li>• The application must allocate the data that <i>ptr</i> points to. DB-Library never manipulates this data; it merely saves the pointer to it for later use by the application.</li> <li>• Here is an example of using this routine to handle deadlock, a situation which occurs occasionally in high-volume applications. See the <i>Adaptive Server Enterprise System Administration Guide</i>. This program fragment sends updates to the server. It reruns the transaction when its message handler detects deadlock.</li> </ul> |

```

...
/*
** Deadlock detection:
**   In the DBPROCESS structure, we save a pointer to
**   a DBBOOL variable. The message handler sets the
**   variable when deadlock occurs. The result
**   processing logic checks the variable and resends
**   the transaction in case of deadlock.
*/

/*
** Allocate the space for the DBBOOL variable
** and save it in the DBPROCESS structure.
*/
    dbsetuserdata(dbproc, malloc(sizeof(DBBOOL)));

    /* Initialize the variable to FALSE */

```

```
        *((DBBOOL *) dbgetuserdata(dbproc)) = FALSE;
        ...
        /* Run queries and check for deadlock */
deadlock:
    /*
    ** Did we get here using deadlock?
    ** If so, the server has already aborted the
    ** transaction. We'll just start it again. In a
    ** real application, the deadlock handling may need
    ** to be somewhat more sophisticated. For
    ** instance, you may want to keep a counter and
    ** retry the transaction just a fixed number
    ** of times.
    */
    if (*((DBBOOL *) dbgetuserdata(dbproc)) == TRUE)
    {
        /* Reset the variable to FALSE */
        *((DBBOOL *) dbgetuserdata(dbproc)) = FALSE;
    }
    /* Start the transaction */
    dbcmd(dbproc, "begin transaction ");
    /* Run the first update command */
    dbcmd(dbproc, "update .....");
    dbsqlxexec(dbproc);
    while (dbresults(dbproc) != NO_MORE_RESULTS)
    {
        /* application code */
    }
    /* Did we deadlock? */
    if (*((DBBOOL *) dbgetuserdata(dbproc)) == TRUE)
        goto deadlock;
    /* Run the second update command. */
    dbcmd(dbproc, "update .....");
    dbsqlxexec(dbproc);
    while (dbresults(dbproc) != NO_MORE_RESULTS)
    {
        /* application code */
    }
    /* Did we deadlock? */
    if (*((DBBOOL *) dbgetuserdata(dbproc)) == TRUE)
        goto deadlock;
    /* No deadlock -- Commit the transaction */
    dbcmd(dbproc, "commit transaction");
    dbsqlxexec(dbproc);
    dbresults(dbproc);
    ...
```

```

/*
** SERVERMSG
** This is the server message handler. Assume that
** the dbmsghandle() routine installed it earlier in
** the program.
*/
servermsgs(dbproc, msgno, msgstate, severity, msgtext,
           srvname, procname, line)
DBPROCESS   *dbproc;
DBINT       msgno;
int         msgstate;
int         severity;
char        *msgtext;
char        *srvname;
char        *procname;
DBUSMALLINT line;
{
    /* Is this a deadlock message? */
    if (msgno == 1205)
    {
        /* Set the deadlock indicator */
        *((DBBOOL *) dbgetuserdata(dbproc)) = TRUE;
        return (0);
    }
    /* Normal message handling code here */
}

```

See also `dbgetuserdata`

## dbsetversion

|             |   |
|-------------|---|
| Description | Specify a DB-Library version level.   |
| Syntax      | RETCODE dbsetversion(version)   |
|             | DBINT      version;   |
| Parameters  | version   |
|             | The version of DB-Library behavior that the application expects. Table 2-29 lists the symbolic values that are legal for <i>version</i> . |

Table 2-29: Values for version (dbsetversion)

| Value of version | Indicates     | Features supported   |
|------------------|---------------|--|
| DBVERSION_46     | 4.6 behavior  | RPCs, registered procedures, remote procedure calls, text and image datatypes.<br>This is the default version of DB-Library. |
| DBVERSION_100    | 10.0 behavior | numeric and decimal datatypes.   |

Return value

SUCCEED or FAIL.

Usage

- dbsetversion sets the version of DB-Library behavior that an application expects. DB-Library will provide the behavior requested, regardless of the actual version of DB-Library in use.
- An application is not required to call dbsetversion. However, if dbsetversion is not called, DB-Library provides version 4.6-level behavior.
- If an application calls dbsetversion, it must do so before calling any other DB-Library routine, with the exception of dbinit.
- If you call dbsetversion more than once, an error occurs.

Note

- You can set the DB-Library version level at runtime using the SYBOCS\_DBVERSION environment variable. When set, this variable changes the application code to use the DB-Library value stored in this variable as the version level.
- If this environment variable is not defined, DB-Library provides 4.6-level behavior or uses the version level requested by an explicit dbsetversion call. If the environment variable is defined and dbsetversion is also called, the dbsetversion overrides the environment variable.

See also

dbinit

dbspid

Description

Get the server process ID for the specified DBPROCESS.

Syntax

```
int dbspid(dbproc)

DBPROCESS *dbproc;
```



|              |   |
|--------------|---|
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>               |
| Return value | <p><i>dbproc</i>'s server process ID.</p>   |
| Usage        | <ul style="list-style-type: none"><li>• <b>dbspid</b> yields the server process ID of the specified DBPROCESS. The process ID appears in the server's sysprocesses table.</li><li>• You can use the server process ID to make queries against the sysprocesses table.</li></ul> |
| See also     | <p><b>dbopen</b></p>  |

## dbspr1row

|              |   |
|--------------|---|
| Description  | <p>Place one row of server query results into a buffer.</p>   |
| Syntax       | <p>RETCODE <b>dbspr1row</b>(<b>dbproc</b>, <b>buffer</b>, <b>buf_len</b>)</p> <pre>DBPROCESS  *dbproc;<br/>char       *buffer;<br/>DBINT      buf_len;</pre>  |
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>buffer</b></p> <p>A pointer to a character buffer to contain the <b>dbspr1row</b> results.</p> <p><b>buf_len</b></p> <p>The length of <i>buffer</i>, including its null terminator.</p> |
| Return value | <p>SUCCEED or FAIL.</p> <hr/> <p><b>Note</b> If an error occurs, the contents of <i>*buffer</i> are undefined.</p> <hr/>  |
| Usage        | <ul style="list-style-type: none"><li>• <b>dbspr1row</b> fills a programmer-supplied buffer with a null-terminated character string containing one server query results row.</li></ul>  |

- `dbspr1row` is useful when displaying data for debugging and writing applications that scroll data displays.
- `dbspr1row` gives programmers greater control over data display than `dbprrow`. `dbprrow` always writes its output to the display device, while `dbspr1row` writes its output to a buffer, which the programmer may then display at whatever time or location is desired.
- To pad results data to its maximum converted length, specify a pad character through the DB-Library option `DBPRPAD`. The pad character will be appended to each column's data. The maximum converted column length is equal to the longest possible string that could be the column's displayable data, or the length of the column's name, whichever is greater. See Options on page 407 for more details on the `DBPRPAD` option.
- You can specify the column separator string using the DB-Library option `DBPRCOLSEP`. The column separator will be added to the end of each converted column's data except the last. The default separator is an ASCII 0x20 (space). See Options on page 407 for more details on the `DBPRCOLSEP` option.
- You can specify the maximum number of characters to be placed on one line using the DB-Library option `DBPRLINELEN`.
- You can specify the line separator string using the DB-Library option `DBPRLINESEP`. The default line separator is a new line (ASCII 0x0a or 0x0d, depending on the host system). See Options on page 407 for more details on the `DBPRLINELEN` and `DBPRLINESEP` options.
- The length of the buffer required by `dbspr1row` can be determined by calling `dbspr1rowlen`.
- The format of results rows returned by `dbspr1row` is determined by the SQL query. `dbspr1row` makes no attempt to format the data beyond converting it to printable characters, padding the columns as necessary, and adding the column and line separators.
- To make the best use of `dbspr1row`, application programs should call it once for every successful call to `dbnextrow`.
- The following code fragment illustrates the use of `dbspr1row`:

```
char      mybuffer[2000];

while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
    dbspr1row(dbproc, mybuffer, sizeof(mybuffer));
    fprintf( stdout, "\n%s", mybuffer);
}
```

- The following code fragment shows the use of the DBPRPAD and DBPRCOLSEP options:

```
char      mybuffer[2000];

/*
** Specify the pad and column separator
** characters */

/* Pad = 0x2A */
dbsetopt(dbproc, DBPRPAD, "*", DBPADON);
/* Col. sep. = 0x2C20 */
dbsetopt(dbproc, DBPRCOLSEP, ", ", 2);

while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
    dbspr1row(dbproc, mybuffer,
              sizeof(mybuffer) );
    fprintf( stdout, "\n%s", mybuffer);
}

/* Turn padding off */
dbsetopt(dbproc, DBPRPAD, SS, DBPADOFF );
/* Revert to default */
dbsetopt(dbproc, DBPRCOLSEP, RS, -1 );
```

See also

dbclropt, dbisopt, dbprhead, dbprrow, dbspr1rowlen, dbsprhead, dbsprline,  
Options on page 407

## dbspr1rowlen

|             |   |
|-------------|---|
| Description | Determine how large a buffer to allocate to hold the results returned by<br>dbsprhead, dbsprline, and dbspr1row.  |
| Syntax      | DBINT dbspr1rowlen(dbproc)  |
|             | DBPROCESS *dbproc;  |
| Parameters  | dbproc<br>A pointer to the DBPROCESS structure that provides the connection for a<br>particular front-end/server process. It contains all the information that DB-<br>Library uses to manage communications and data between the front end and<br>server. |

|              |   |
|--------------|---|
| Return value | The size of the buffer, in bytes, required by <code>dbsprhead</code> , <code>dbsprline</code> , and <code>dbspr1row</code> on success; a negative integer on error.   |
| Usage        | <ul style="list-style-type: none"> <li>• <code>dbspr1rowlen</code> determines the size of the buffer (in bytes) required by <code>dbsprhead</code>, <code>dbsprline</code>, and <code>dbspr1row</code>, including the null terminator.</li> <li>• <code>dbspr1rowlen</code> is useful when printing data for debugging and when scrolling data displays.</li> <li>• To make the best use of <code>dbspr1rowlen</code>, application programs should call it once for every successful call to <code>dbresults</code>.</li> <li>• The following code fragment illustrates the use of <code>dbspr1rowlen</code>: <pre> dbcmd(dbproc, "select * from sysdatabases"); dbcmd(dbproc, " order by name"); dbcmd(dbproc, " compute max(crdate) by name");  dbsqlexec(dbproc); dbresults(dbproc); printf("Maximum row length will be %ld \ characters.\n", dspr1rowlen(dbproc)); </pre> </li> </ul> |
| See also     | <code>dbprhead</code> , <code>dbprrow</code> , <code>dbspr1row</code> , <code>dbsprhead</code> , <code>dbsprline</code> , Options on page 407   |

## dbsprhead

|             |   |
|-------------|---|
| Description | Place the server query results header into a buffer.  |
| Syntax      | <pre> RETCODE dsprhead(dbproc, buffer, buf_len)  DBPROCESS  *dbproc; char        *buffer; DBINT       buf_len; </pre>   |
| Parameters  | <p><b>dbproc</b><br/>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>buffer</b><br/>A pointer to a character buffer to contain the query results header.</p> <p><b>buf_len</b><br/>The length of <i>buffer</i>, including its null terminator.</p> |

Return value                    SUCCEED or FAIL.

---

**Note** If an error occurs, the contents of *\*buffer* are undefined.

---

## Usage

- `dbsprhead` fills a programmer-supplied buffer with a null-terminated character string containing the header for the current set of query results. The header consists of the column names. The sequence of the column names matches that of the output of `dbspr1row`.
- `dbsprhead` is useful when printing data for debugging, and when scrolling data displays.
- To pad each column name to its maximum converted length, specify a pad character using the DB-Library option `DBPRPAD`. The pad character will be appended to each column's name. The maximum converted column length is equal to the longest possible string that could be the column's displayable data, or the length of the column's name, whichever is greater. See Options on page 407 for more details on the `DBPRPAD` option.
- You can specify the column separator string using the DB-Library option `DBPRCOLSEP`. The column separator will be added to the end of each column name except the last. The default separator is an ASCII 0x20 (space). See Options on page 407 for more details on the `DBPRCOLSEP` option.

- You can specify the maximum number of characters to be placed on one line using the DB-Library option `DBPRLINELEN`.

You can specify the line separator string using the DB-Library option `DBPRLINESEP`. The default line separator is a newline (ASCII 0x0a or 0x0d, depending on the host system). See Options on page 407 for more details on the `DBPRLINELEN` and `DBPRLINESEP` options.

- The length of the buffer required by `dbsprhead` can be determined by calling `dbspr1rowlen`.
- To make the best use of `dbsprhead`, application programs should call it once for every successful call to `dbresults`.
- The following code fragment illustrates the use of `dbsprhead`:

```
dbcmd(dbproc, "select * from sysdatabases");
dbcmd(dbproc, " order by name");
dbcmd(dbproc, " compute max(crdate) by name");

dbsqlxexec(dbproc);
dbresults(dbproc);
```

```
    dbsprhead(dbproc, buffer, sizeof(buffer));
    printf("%s\n", buffer);
```

See also `dbprhead`, `dbprrow`, `dbsetopt`, `dbsprlrow`, `dbsprlrowlen`, `dbsprline`, Options on page 407

## dbsprline

|              |   |
|--------------|---|
| Description  | Get a formatted string that contains underlining for the column names produced by <code>dbsprhead</code> .  |
| Syntax       | <pre>RETCODE dbsprline(dbproc, buffer, buf_len, linechar)</pre> <p> <b>DBPROCESS</b>    <i>*dbproc;</i><br/> <b>char</b>            <i>*buffer;</i><br/> <b>DBINT</b>           <i>buf_len;</i><br/> <b>DBCHAR</b>          <i>linechar;</i> </p>   |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the <b>DBPROCESS</b> structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>buffer</b><br/>A pointer to a character buffer to contain the <code>dbsprline</code> results.</p> <p><b>buf_len</b><br/>The length of <i>buffer</i>, including its null terminator.</p> <p><b>linechar</b><br/>The character with which to “underline” column names produced by <code>dbsprhead</code>.</p> |
| Return value | SUCCESS or FAIL.  |

---

**Note** If an error occurs, the contents of *\*buffer* are undefined.

---

- Usage
- `dbsprline` is used to “underline” the column names produced by `dbsprhead`. `dbsprline` fills a programmer-supplied buffer with a null-terminated character string containing one group of the character specified by *linechar* for each column in the current set of query results. The format of this line matches the format of the output of `dbsprhead`.

- You can determine the length of the buffer required by `dbsprline` using `dbspr1rowlen`.
- To make the best use of `dbsprhead`, application programs should call it once for every successful call to `dbresults`.
- `dbsprline` is useful when printing data for debugging, and when scrolling data displays.
- The following code fragment illustrates the use of `dbsprline`:

```

dbcmd(dbproc, "select * from sysdatabases");
dbcmd(dbproc, " order by name");
dbcmd(dbproc, " compute max(crdate) by name");

dbsqlexec(dbproc);
dbresults(dbproc);

/*
** Display the column headings, underline them
** with "*"
*/
dbsprhead(dbproc, buffer, sizeof(buffer));
printf("%s\n", buffer);

dbsprline(dbproc, buffer, sizeof(buffer), '*');
printf("%s\n", buffer);

/* Process returned rows as usual */

```

See also

`dbprhead`, `dbprrow`, `dbspr1row`, `dbspr1rowlen`, `dbsprhead`, Options on page 407

## dbsqlexec

Description

Send a command batch to the server.

Syntax

`RETCODE dbsqlexec(dbproc)`

`DBPROCESS *dbproc;`

Parameters

`dbproc`

A pointer to the `DBPROCESS` structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

**Return value**

SUCCEED or FAIL.

The most common reason for failing is a SQL syntax error. dbsqlexec will also fail if there are semantic errors, such as incorrect column or table names. Failure occurs if any of the commands in the batch contains a semantic or syntax error. dbsqlexec also fails if previous results had not been processed, or if the command buffer was empty.

In addition, a runtime error, such as a database protection violation, can cause dbsqlexec to fail. A runtime error will cause dbsqlexec to fail:

- If the command causing the error is the only command in the command buffer
- If the command causing the error is the first command in a multiple-command buffer

If the command buffer contains multiple commands (and the first command in the buffer is ok), a runtime error will not cause dbsqlexec to fail. Instead, failure will occur with the dbresults call that processes the command causing the runtime error.

The situation is a bit more complicated for runtime errors and stored procedures. A runtime error on an execute command may cause dbsqlexec to fail, in accordance with the rule given in the previous paragraphs. A runtime error on a statement *inside* a stored procedure will not cause dbsqlexec to fail, however. For example, if the stored procedure contains an insert statement and the user does not have insert permission on the database table, the insert statement fails, but dbsqlexec will still return SUCCEED. To check for runtime errors inside stored procedures, use the dbretstatus routine to look at the procedure's return status, and trap relevant server messages inside your message handler.

**Usage**

- This routine sends SQL commands, stored in the command buffer of the DBPROCESS, to the server. Commands may be added to the DBPROCESS structure by calling dbcmd or dbcmd.
- Once dbsqlexec returns SUCCEED, the application must call dbresults to process the results.
- The typical sequence of calls is:

```
DBINT      xvariable;  
DBCHAR     yvariable[10];  
  
/* Read the query into the command buffer */  
dbcmd(dbproc, "select x = 100, y = 'hello'");  
  
/* Send the query to Adaptive Server Enterprise */
```



```

    dbsqlxexec(dbproc);

/* Get ready to process the query results */
dbresults(dbproc);

/* Bind column data to program variables */
dbbind(dbproc, 1, INTBIND, (DBINT) 0,
        (BYTE *) &xvariable);
dbbind(dbproc, 2, STRINGBIND, (DBINT) 0,
        yvariable);

/* Now process each row */
while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
    C-code to print or process row data
}

```

- `dbsqlxexec` is equivalent to `dbsqlsend` followed by `dbsqllok`. However, after sending a query to the server, `dbsqlxexec` waits until a response is received or until the timeout period has elapsed. By substituting `dbsqlsend` and `dbsqllok` for `dbsqlxexec`, you can sometimes provide a way for the application to respond more effectively to multiple input and output streams. See the reference pages for `dbsqlsend` and `dbsqllok`.
- Multiple commands may exist in the command buffer when an application calls `dbsqlxexec`. These commands are sent to the server as a unit and are considered to be a single command batch.

See also

`dbcmd`, `dbfcmd`, `dbnextrow`, `dbresults`, `dbretstatus`, `dbsettime`, `dbsqllok`, `dbsqlsend`

## dbsqllok

Description

Wait for results from the server and verify the correctness of the instructions the server is responding to.

Syntax

`RETCODE dbsqllok(dbproc)`

Parameters

`dbproc`

A pointer to the `DBPROCESS` structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

## Return value

SUCCEED or FAIL.

The most common reason for failing is a SQL syntax error. dbsqllok will also fail if there are semantic errors, such as incorrect column or table names. Failure occurs if any of the commands in the batch contains a semantic or syntax error.

In addition, a runtime error, such as a database protection violation, will cause dbsqllok to fail *if* the command buffer contains only a single command. If the command buffer contains multiple commands, a runtime error will *not* cause dbsqllok to fail. Instead, failure will occur with the dbresults call that processes the command causing the runtime error.

The situation is a bit more complicated for runtime errors and stored procedures. A runtime error on an execute command may cause dbsqllok to fail, in accordance with the rule given in the previous paragraph. A runtime error on a statement *inside* a stored procedure will not cause dbsqllok to fail, however. For example, if the stored procedure contains an insert statement and the user does not have insert permission on the database table, the insert statement fails, but dbsqllok will still return SUCCEED. To check for runtime errors inside stored procedures, use the dbretstatus routine to look at the procedure's return status and trap relevant server messages inside your message handler.

## Usage

- dbsqllok reports the success or failure of a server command and initiates results processing for successful commands.
- A successful dbsqllok call must always be followed by a call to dbresults to process the results.
- dbsqllok is useful in the following situations:
  - After a dbsqlsend call  
dbsqllok must be called after a batch of Transact-SQL commands is sent to the server with dbsqlsend.
  - After a dbrpcsend call  
dbsqllok must be called after an RPC command is sent with dbrpcinit, dbrpcparam, and dbrpcsend.
  - After calls to dbwritetext or dbmoretext  
dbsqllok must be called after a text update command is sent to the server by a call to dbwritetext or dbmoretext.

Using dbsqllok with dbsqlsend

- `dbsqllok` initiates results processing after a call to `dbsqlsend`.
- `dbsqllok` and `dbsqlsend` provide an alternative to `dbsqlexec`. `dbsqlexec` sends a command batch and waits for initial results from the server. The application is blocked from doing anything else until results arrive. When `dbsqlsend` and `dbsqllok` are used with `dbpoll`, the application has a non-blocking alternative. The typical control sequence is as follows:
  - A call to `dbsqlsend` sends the command to the server.
  - The program calls `dbpoll` in a loop to check for the arrival of server results. Non-related work can be performed during each loop iteration. The loop terminates when `dbpoll` indicates results have arrived.
  - A call to `dbsqllok` reports success or failure and initiates results processing if successful.

---

**Note** On occasion, `dbpoll` may report that data is ready for `dbsqllok` to read when only the first bytes of the server response are present. When this occurs, `dbsqllok` waits for the rest of the response or until the timeout period has elapsed, just like `dbsqlexec`. In practice, however, the entire response is usually available at one time.

---

- The example below illustrates the use of `dbsqllok` and `dbpoll`. The example calls an application function, `busy_wait`, to execute a `dbpoll` loop. Here is the mainline code that calls `busy_wait`:

```
/*
** This is a query that will take some time.
*/
dbcmd(dbproc, "waitfor delay '00:00:05' select its = 'over'");

/*
** Send the query with dbsqlsend. dbsqlsend does not
** wait for a server response.
*/
retcode = dbsqlsend(dbproc);
if (retcode != SUCCEED)
{
    fprintf(stdout, "dbsqlsend failed. Exiting.\n");
    dbexit();
    exit(ERREXIT);
}

/*
```

```
/** If we call dbsqllok() now, it might block. But, we can use
** a dbpoll() loop to get some other work done while
** we are waiting for the results.
**/
busy_wait(dbproc);

/*
** Now there should be some results waiting to be read, so
** call dbsqllok().
**/
retcode = dbsqllok(dbproc);
if (retcode != SUCCEED)
{
    fprintf(stdout, "Query failed.\n");
}
else
{
    ... dbresults() loop goes here ...
}
```

busy\_wait executes a dbpoll loop. During each iteration of the loop, a call to dbpoll determines whether results have arrived. If results have arrived, busy\_wait returns. Otherwise, the function wait\_work is called. wait\_work performs a piece of non-related work, then returns. The functions wait\_work\_init and wait\_work\_cleanup perform initialization and cleanup for wait\_work. Here is the code for these functions:

```
void busy_wait(dbproc)
DBPROCESS *dbproc;
{
    RETCODE retcode;
    DBPROCESS *ready_dbproc;
    int poll_ret_reason;

    wait_work_init();
    while(1)
    {
        retcode = dbpoll(dbproc, 0, &ready_dbproc, &poll_ret_reason);
        if (retcode != SUCCEED)
        {
            fprintf(stdout, "dbpoll() failed! Exiting.\n");
            dbexit();
            exit(ERREXIT);
        }
        if (poll_ret_reason == DBRESULT)
        {
            /*
```

```

        ** Query results have arrived. Now we break out of
        ** the loop and return. Our caller can then call dbsqllok().
        */
        break; /* while */
    }
    else
    {
        /*
        ** Here's where we can do some non-related work while we
        ** are waiting.
        */
        wait_work();
    }
} /* while */
wait_work_cleanup();
} /* busy_wait */

/* These globals are used by the wait functions. */
static int wait_pos;
static char wait_char;
void wait_work()
{
    /*
    ** "work", as defined here, consists of drawing a 'w' or 'W' to
    ** the terminal. We output one character each time we are called.
    ** When we reach the 65th character position, we switch from
    ** 'w' to 'W' (or vice-versa) and start over.
    */
    fputc(wait_char, stdout);
    ++wait_pos;
    if (wait_pos >= 65)
    {
        /*
        ** Go back to the beginning of the line, then switch from
        ** 'W' to 'w' or vice versa.
        */
        fputc('\r', stdout);
        wait_pos = 0;
        wait_char = (wait_char == 'w' ? 'W' : 'w');
    }
}
void wait_work_init()
{
    wait_pos = 0;
    wait_char = 'w';
}

```

```
}  
void wait_work_cleanup()  
{  
    fputc('\n', stdout);  
}
```

#### Using dbsqllok with dbrpcsend

- dbsqllok initiates results processing after an RPC command. RPC commands are constructed and sent with dbrpcinit, dbrpcparam, and dbrpcsend. After dbrpcsend, the program must call dbsqllok.
- dbpoll can be called in a loop to poll for a server response between dbrpcsend and dbsqllok.
- See the reference pages for dbrpcinit, dbrpcparam, and dbrpcsend. The sample program *example8.c* demonstrates an RPC command.

#### Using dbsqllok with dbwritetext and dbmoretext

- dbsqllok initiates results processing after a text update command. For text updates, chunks of text can be sent to the server with dbwritetext and dbmoretext. After both of these calls, dbsqllok must be called.
- See the reference pages for dbwritetext and dbmoretext. dbwritetext has an example.

See also

dbcmd, dbfcmd, DBIORDESC, DBIOWDESC, dbmoretext, dbnextrow, dbpoll, DBRBUF, dbresults, dbretstatus, dbrpcsend, dbsettime, dbsqlexec, dbsqlsend, dbwritetext

## dbsqlsend

Description

Send a command batch to the server and do not wait for a response.

Syntax

RETCODE dbsqlsend(dbproc)

DBPROCESS \*dbproc;

|              |   |
|--------------|---|
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>   |
| Return value | <p>SUCCEED or FAIL.</p> <p>dbsqlsend may fail if previous results had not been processed, or if the command buffer was empty.</p>   |
| Usage        | <ul style="list-style-type: none"><li>• This routine sends SQL commands, stored in the command buffer, to the server. The application can add commands to the command buffer by calling dbcmd or dbcmd.</li><li>• Once dbsqlsend returns SUCCEED, the application must call dbsqlok to verify the accuracy of the command batch. The application can then call dbresults to process the results.</li><li>• dbsqlxec is equivalent to dbsqlsend followed by dbsqlok.</li><li>• The use of dbsqlsend with dbsqlok is of particular value in UNIX applications. After sending a query to the server, dbsqlxec waits until a response is received or until the timeout period has elapsed. By substituting dbsqlsend, dbpoll and dbsqlok for dbsqlxec, you can sometimes provide a way for an application to respond more effectively to multiple input and output streams. See the dbsqlok reference page.</li></ul> |
| See also     | <p>dbcmd, dbcmd, DBIORDESC, DBIOWDESC, dbnextrow, dbpoll, dbresults, dbsettime, dbsqlxec, dbsqlok</p>   |

## dbstrbuild

|             |   |
|-------------|---|
| Description | Build a printable string from text containing placeholders for variables.   |
| Syntax      | <pre>int dbstrbuild(dbproc, charbuf, bufsize,<br/>              text [, formats [, arg] ... ])</pre><br><pre>DBPROCESS  *dbproc;<br/>char       *charbuf;<br/>int        bufsize;<br/>char       *text;<br/>char       *formats;<br/>???       args...;</pre> |

## Parameters

## dbproc

A pointer to the DBPROCESS that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server. dbstrbuild uses it only as a parameter to the programmer-installed error handler (if one exists) when an error occurs.

## charbuf

A pointer to the destination buffer that will contain the message built by dbstrbuild.

## bufsize

The size of the destination buffer, in bytes. This size must include a single byte for the results string's null terminator.

## text

A pointer to a null-terminated character string that contains message text and placeholders for variables. Placeholders consist of a percent sign, an integer, and an exclamation point. The integer indicates which argument to substitute for a particular placeholder. Arguments and format strings are numbered from left to right. Argument 1 is substituted for placeholder “%1!”, and so on.

## formats

A pointer to a null-terminated string containing one sprintf-style format specifier for each place holder in the *text* string.

## args

The values that will be converted according to the contents of the *formats* string. There must be one argument for each format in the *formats* string. The first value will correspond to the “%1!” parameter, the second the “%2!”, and so forth. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments are simply ignored.

## Return value

On success, the length of the resulting message string, not including the null terminator; on failure, a negative integer.

## Usage

- Parameters in error messages can occur in different orders in different languages. dbstrbuild allows construction of error messages in a manner similar to the C standard-library sprintf routine. Use of dbstrbuild ensures easy translation of error messages from one language to another.
- dbstrbuild builds a printable string from an error text that contains placeholders for variables, a format string containing information about the types and appearances of those variables, and a variable number of arguments that provide actual values for those variables.



- Placeholders for variables consist of a percent sign, an integer, and an exclamation point. The integer indicates which argument to substitute for a particular placeholder. Arguments and format strings are numbered from left to right. Argument 1 is substituted for placeholder “%1!”, and so on.

For example, consider an error message that complains about a misused keyword in a stored procedure. The message requires three arguments: the misused keyword, the line in which the keyword occurs, and the name of the stored procedure in which the misuse occurs. In the English localization file, the message text might appear as:

```
The keyword '%1!' is misused in line %2! of stored
procedure '%3!' .
```

In the localization file, the same message might appear as:

```
In line '%2!' of stored procedure '%3!', the keyword
'%1!' misused is.
```

The `dbstrbuild` line for either of the above messages would be:

```
dbstrbuild(dbproc, charbuf, BUFSIZE, <get the
message somehow>, "%s %d %s", keyword,
linenum, sp_name)
```

*keyword* is substituted for placeholder “%1!”, *linenum* is substituted for placeholder “%2!”, and *sp\_name* is substituted for placeholder “%3!”.

- The following code fragment illustrates the use of `dbstrbuild` to build messages. For simplicity, the text of the message is hard-coded. In practice, `dbstrbuild` message texts come from a localization file.

```
char    charbuf[BUFSIZE];
int     linenum = 15;
char    *filename = "myfile";
char    *dirname = "mydir";

dbstrbuild (dbproc, charbuf, BUFSIZE,
"Unable to read line %1! of file %2! in \
directory %3!.", "%d %s %s", linenum,
filename, dirname);
printf(charbuf);
```

- `dbstrbuild` format specifiers may be separated by any other characters, or they may be adjacent to each other. This allows pre-existing English-language message strings to be used as `dbstrbuild` format parameters. The first format specifier describes the “%1!” parameter, the second the “%2!” parameter, and so forth.

See also

`dbconvert`, `dbdatetime`, `dbdatepart`

## dbstrcmp

**Description** Compares two character strings using a specified sort order.

**Syntax** `int dbstrcmp(dbproc, str1, len1, str2, len2, sortorder)`

```
DBPROCESS    *dbproc;
char          *str1;
int           len1;
char          *str2;
int           len2;
DBSORTORDER  *sortorder;
```

**Parameters**

**dbproc**

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

**str1**

A pointer to the first character string to compare. *str1* may be NULL.

**len1**

The length, in bytes, of *str1*. If *len1* is -1, *str1* is assumed to be null-terminated.

**str2**

A pointer to the second character string to compare. *str2* may be NULL.

**len2**

The length, in bytes, of *str2*. If *len2* is -1, *str2* is assumed to be null-terminated.

**sortorder**

A pointer to a DBSORTORDER structure allocated using dbloadsort. If *sortorder* is NULL, dbstrcmp compares *str1* and *str2* using their binary values, just as strcmp does.

**Return value**

- 1 if *str1* is lexicographically greater than *str2*.
- 0 if *str1* is lexicographically equal to *str2*.
- -1 if *str1* is lexicographically less than *str2*.

**Usage**

- dbstrcmp compares *str1* and *str2* and returns an integer greater than, equal to, or less than 0, according to whether *str1* is lexicographically greater than, equal to, or less than *str2*.

- `dbstrcmp` uses a sort order that was retrieved from the server using `dbloadsort`. This allows DB-Library application programs to compare strings using the same sort order as the server.
- Note that some languages contain strings that are lexicographically equal according to some specified sort order, but contain different characters. Even though they are “equal,” there is a standard order that should be used when placing them into an ordered list. When given two strings like this to compare, `dbstrcmp` returns 0 (indicating the two strings are equal), but `dbstrsort` returns some non-zero value indicating that one of these strings should appear before the other in a sorted list.

Below is an example of this behavior. The two English-language character strings are used with a case-insensitive sort order that specifies that uppercase letters should appear before lowercase:

```
/* This call returns 0: */
dbstrcmp(dbproc, "ABC", 3, "abc", 3, mysort);

/* This call returns a negative value: */
dbstrsort(dbproc, "ABC", 3, "abc", 3, mysort);
```

See also

`dbfreesort`, `dbloadsort`, `dbstrsort`

## dbstrcpy

Description

Copy all or a portion of the command buffer.

Syntax

```
RETCODE dbstrcpy(dbproc, start, numbytes, dest)
```

```
DBPROCESS  *dbproc;
int         start;
int         numbytes;
char        *dest;
```

Parameters

`dbproc`

A pointer to the `DBPROCESS` structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

`start`

Character position in the command buffer to start copying from. The first character has position 0. If *start* is greater than the length of the command buffer, `dbstrcpy` inserts a null terminator at *dest[0]*.

**numbytes**

The number of characters to copy. If *numbytes* is -1, dbstrcpy will copy the entire command buffer, whether or not *dest* points to adequate space. It is legal to copy 0 bytes, in which case dbstrcpy inserts a null terminator at *dest[0]*. If there are not *numbytes* available to copy, dbstrcpy copies the number of bytes available and returns SUCCEED.

**dest**

A pointer to the destination buffer to copy the source string into. Before calling dbstrcpy, the caller must verify that the destination buffer is large enough to hold the copied characters. The function dbstrlen returns the size of the entire command buffer.

**Return value**

SUCCEED or FAIL.

dbstrcpy returns FAIL if *start* is negative.

**Usage**

- dbstrcpy copies a portion of the command buffer to a string buffer supplied by the application. The copy is null-terminated.
- Internally, the command buffer is a linked list of non-null-terminated text strings. dbgetchar, dbstrcpy, and dbstrlen together provide a way to locate and copy parts of the command buffer.
- dbstrcpy assumes that the destination is large enough to receive the source string. If not, a segmentation fault is likely.
- When *numbytes* is passed as -1, dbstrcpy copies the entire command buffer. Do not pass *numbytes* as -1 unless you are certain that *dest* points to adequate space for this string. The function dbstrlen returns the length of the current command string.
- The following fragment shows how to print the entire command buffer to a file:

```
FILE      *outfile;
DBPROCESS *dbproc;
char      *prbuf; /* buffer for collecting the command buffer
                  ** contents as a null-terminated string
                  */
RETCODE return_code;

/*
** Allocate sufficient space. dbstrlen() returns the number of
** characters currently in the command buffer. We need one
** more byte because dbstrcpy will append a null terminator.
** NOTE that memory allocation and disposal may be done
** differently on your platform.
**/
```

```

prbuf = (char *) malloc(dbstrlen(dbproc) + 1);
if (prbuf == NULL)
{
    fprintf(stderr, "Out of memory.");
    dbexit();
    exit(ERREXIT); /* ERREXIT is defined in the DB-lib headers */
}
/* Copy the command buffer into the allocated space: */
return_code = dbstrcpy(dbproc, 0, -1, prbuf);
assert(return_code == SUCCEED);

/* Print the contents: */
fprintf(outfile, "%s", prbuf);

/* Free the buffer: */
free(prbuf);

```

See also `dbcmd`, `dbfcmd`, `dbfreebuf`, `dbgetchar`, `dbstrlen`

## dbstrlen

|              |  |
|--------------|--|
| Description  | Return the length, in characters, of the command buffer.   |
| Syntax       | <pre>int dbstrlen(dbproc)</pre> <p>DBPROCESS *dbproc;</p>  |
| Parameters   | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>   |
| Return value | The length, in characters, of the command buffer.  |
| Usage        | <ul style="list-style-type: none"> <li>• <code>dbstrlen</code> returns the length, in characters, of the SQL command text in the command buffer.</li> <li>• Internally, the command buffer is a linked list of non-null-terminated text strings. <code>dbgetchar</code>, <code>dbstrcpy</code>, and <code>dbstrlen</code> together provide a way to locate and copy parts of the command buffer.</li> <li>• Before you copy the command buffer with <code>dbstrcpy</code>, use <code>dbstrlen</code> to make sure that the destination buffer is large enough.</li> <li>• The count returned by <code>dbstrlen</code> does not include space for a null terminator.</li> </ul> |

See also dbcmd, dbfcmd, dbfreebuf, dbgetchar, dbstrcpy

## dbstrsort

**Description** Determine which of two character strings should appear first in a sorted list.

**Syntax** `int dbstrsort(dbproc, str1, len1, str2, len2,  
sortorder)`

```
DBPROCESS    *dbproc;  
char          *str1;  
int           len1;  
char          *str2;  
int           len2;  
DBSORTORDER  *sortorder;
```

**Parameters**

**dbproc**

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

**str1**

A pointer to the first character string to compare. *str1* may be NULL.

**len1**

The length, in bytes, of *str1*. If *len1* is -1, *str1* is assumed to be null-terminated.

**str2**

A pointer to the second character string to compare. *str2* may be NULL.

**len2**

The length, in bytes, of *str2*. If *len2* is -1, *str2* is assumed to be null-terminated.

**sortorder**

A pointer to a DBSORTORDER structure allocated using dbloadsort. If *sortorder* is NULL, dbstrsort compares *str1* and *str2* using their binary values, just as strcmp does.

**Return value**

- 1 if *str1* should appear after *str2*.
- 0 if *str1* is identical to *str2*.
- -1 if *str1* should appear before *str2*.

## Usage

- `dbstrsort` compares *str1* and *str2* and returns an integer greater than, equal to, or less than 0, according to whether *str1* should appear after, at the same place (the strings are identical), or before *str2* in a sorted list.
- `dbstrsort` uses a sort order that was retrieved from the server using `dbloadsort`. This allows DB-Library application programs to compare strings using the same sort order as the server.
- Note that some languages contain strings that are lexicographically equal according to some specified sort order, but contain different characters. Even though they are “equal,” there is a standard order that should be used when placing them into an ordered list. When given two strings like this to compare, `dbstrcmp` returns 0 (indicating the two strings are equal), but `dbstrsort` returns some non-zero value indicating that one of these strings should appear before the other in a sorted list.

Below is an example of this behavior. The two English-language character strings are used with a case-insensitive sort order that specifies that uppercase characters should appear before lowercase:

```
/* This call returns 0: */
dbstrcmp(dbproc, "ABC", 3, "abc", 3, mysort);

/* This call returns a negative value: */
dbstrsort(dbproc, "ABC", 3, "abc", 3, mysort);
```

- `dbstrsort` can only be used to examine two character strings that have already been identified as equal using `dbstrcmp`. If `dbstrcmp` has not identified these strings as being equal to each other, `dbstrsort`'s behavior is undefined.

## See also

`dbfreesort`, `dbloadsort`, `dbstrcmp`

## dbtabbrowse

## Description

Determine whether the specified table is updatable through the DB-Library browse-mode facilities.

## Syntax

DBBOOL dbtabbrowse(dbproc, tabnum)

```
DBPROCESS *dbproc;
int        tabnum;
```

|              |   |
|--------------|---|
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>tabnum</b></p> <p>The number of the table of interest, as specified in the select statement's from clause. Table numbers start at 1.</p>  |
| Return value | "TRUE" or "FALSE".  |
| Usage        | <ul style="list-style-type: none"><li>• dbtabbrowse is one of the DB-Library browse-mode routines. See "Browse mode" on page 26 for a detailed discussion of browse mode.</li><li>• dbtabbrowse provides a way to identify browsable tables. It is useful when examining ad hoc queries prior to performing browse mode updates based on them. If the query has been hard-coded into the program, this routine is obviously unnecessary.</li><li>• For a table to be considered "browsable," it must have a unique index and a timestamp column.</li><li>• The application can call dbtabbrowse anytime after dbresults.</li><li>• The sample program <i>example7.c</i> contains a call to dbtabbrowse.</li></ul> |
| See also     | dbcolbrowse, dbcolsource, dbqual, dbtabcount, dbtabname, dbtabsource, dbtsnewlen, dbtsnewval, dbtsput   |

## dbtabcount

|              |   |
|--------------|---|
| Description  | Return the number of tables involved in the current select query.   |
| Syntax       | <pre>int dbtabcount(dbproc)</pre> <pre>DBPROCESS *dbproc;</pre>   |
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> |
| Return value | The number of tables, including server work tables, involved in the current set of row results.   |



dbtabcount will return -1 in case of error.

#### Usage

- dbtabcount is one of the DB-Library browse-mode routines. It is usable only with results from a browse-mode select (that is, a select containing the key words for browse). See “Browse mode” on page 26 for a detailed discussion of browse mode.
- A select query can generate a set of result rows whose columns are derived from several database tables. To perform browse-mode updates of columns in a query’s select list, the application must know how many tables were involved in the query, because each table requires a separate update statement. dbtabcount can provide this information for ad hoc queries. If the query has been hard-coded into the program, this routine is obviously unnecessary.
- The count returned by this routine includes any server “work tables” used in processing the query. The server sometimes creates temporary, internal work tables to process a query. It deletes these work tables by the time it finishes processing the statement. Work tables are not updatable and are not available to the application. Therefore, before using a table number, the application must make sure that it does not belong to a work table. dbtabname can be used to determine whether a particular table number refers to a work table.
- The application can call dbtabcount anytime after dbresults.
- The sample program *example7.c* contains a call to dbtabcount.

#### See also

dbcolbrowse, dbcolsource, dbqual, dbtabbrowse, dbtabname, dbtabsource, dbtsnewlen, dbtsnewval, dbtspu

## dbtabname

#### Description

Return the name of a table based on its number.

#### Syntax

```
char *dbtabname(dbproc, tabnum)
```

```
DBPROCESS  *dbproc;
int         tabnum;
```

|              |  |
|--------------|--|
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>tabnum</b></p> <p>The number of the table of interest. Table numbers start with 1. Use dbtabcount to find out the total number of tables involved in a particular query.</p>   |
| Return value | <p>A pointer to the null-terminated name of the specified table. This pointer will be NULL if the table number is out of range or if the specified table is a server work table. See the dbtabcount reference page for a description of work tables.</p>   |
| Usage        | <ul style="list-style-type: none"><li>• dbtabname is one of the DB-Library browse-mode routines. It is usable only with results from a browse-mode select (that is, a select containing the key words for browse). See “Browse mode” on page 26 for a detailed discussion of browse mode.</li><li>• A select query can generate a set of result rows whose columns are derived from several database tables. dbtabname provides a way for an application to determine the name of each table involved in an ad hoc query. If the query has been hard-coded into the program, this routine obviously is unnecessary.</li><li>• The application can call dbtabname anytime after dbresults.</li><li>• The sample program <i>example7.c</i> contains a call to dbtabname.</li></ul> |
| See also     | <p>dbcolbrowse, dbcolsource, dbqual, dbtabbrowse, dbtabcount, dbtabsource, dbtsnewlen, dbtsnewval, dbtspu</p>  |

## dbtabsource

|             |  |
|-------------|--|
| Description | <p>Return the name and number of the table from which a particular result column was derived.</p>                  |
| Syntax      | <pre>char *dbtabsource(dbproc, colnum, tabnum)  DBPROCESS  *dbproc; int         colnum; int         *tabnum;</pre> |

|              |  |
|--------------|--|
| Parameters   | <b>dbproc</b><br>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.   |
|              | <b>colnum</b><br>The number of the result column of interest. Column numbers start at 1.   |
|              | <b>tabnum</b><br>A pointer to an integer, which will be filled in with the table's number. Many DB-Library routines that deal with browse mode accept either a table name or a table number. If dbtabsource returns NULL (see the "Returns" section below), <i>*tabnum</i> will be set to -1.  |
| Return value | <p>A pointer to the name of the table from which this result column was derived. A NULL return value can mean a few different things:</p> <ul style="list-style-type: none"><li>• The DBPROCESS is dead or not enabled. This is an error that will cause an application's error handler to be invoked.</li><li>• The column number is out of range.</li><li>• The column is the result of an expression, such as max(colname).</li></ul>   |
| Usage        | <ul style="list-style-type: none"><li>• dbtabsource is one of the DB-Library browse-mode routines. It is usable only with results from a browse-mode select (that is, a select containing the key words for browse). See "Browse mode" on page 26 for a detailed discussion of browse mode.</li><li>• dbtabsource allows an application to determine which tables provided the columns in the current set of result rows. This information is valuable when using dbqual to construct where clauses for update and delete statements based on ad hoc queries. If the query has been hard-coded into the program, this routine obviously is unnecessary.</li><li>• The application can call dbtabsource anytime after dbresults.</li><li>• The sample program <i>example7.c</i> contains a call to dbtabsource.</li></ul> |
| See also     | dbcolbrowse, dbcolsource, dbqual, dbtabbrowse, dbtabcount, dbtabname, dbtsnewlen, dbtsnewval   |

## DBTDS

|              |   |
|--------------|---|
| Description  | Determine which version of TDS (the Tabular Data Stream protocol) is being used.  |
| Syntax       | <pre>int DBTDS(dbproc)</pre><br><pre>DBPROCESS    *dbproc;</pre>  |
| Parameters   | <p><code>dbproc</code></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>   |
| Return value | <p>The version of TDS used by <i>dbproc</i> to communicate with the server. Currently, the possible versions are:</p> <ul style="list-style-type: none"><li>• DBTDS_2_0</li><li>• DBTDS_3_4</li><li>• DBTDS_4_0</li><li>• DBTDS_4_2</li><li>• DBTDS_4_6</li><li>• DBTDS_4_9_5</li><li>• DBTDS_5_0</li></ul> <p>DBTDS returns a negative integer on error.</p> |
| Usage        | <ul style="list-style-type: none"><li>• DBTDS returns the version of TDS (Tabular Data Stream protocol) being used by <i>dbproc</i> to communicate with the server.</li></ul>   |
| See also     | <code>dbversion</code>  |

## dbtextsize

|             |   |
|-------------|---|
| Description | Returns the number of bytes of text or image data that remain to be read for the current row. |
| Syntax      | <pre>DBINT dbtextsize(dbproc)</pre><br><pre>DBPROCESS    *dbproc;</pre>                       |

| Parameters                      | <p><code>dbproc</code></p> <p>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>   |                                 |             |                      |  |                 |                        |                 |   |
|---------------------------------|--|---------------------------------|-------------|----------------------|--|-----------------|------------------------|-----------------|---|
| Return value                    | <p>The following table lists the return values for <code>dbtextsize</code>:</p> <table> <tr> <th><code>dbtextsize</code> returns</th><th>To indicate</th></tr> <tr> <td><code>&gt;= 0</code></td><td>The number of bytes that remain to be read. Zero indicates <code>NO_MORE_ROWS</code>.</td></tr> <tr> <td><code>-1</code></td><td>An error has occurred.</td></tr> <tr> <td><code>-2</code></td><td><code>dbtextsize</code> has been called for RPC data.</td></tr> </table> | <code>dbtextsize</code> returns | To indicate | <code>&gt;= 0</code> | The number of bytes that remain to be read. Zero indicates <code>NO_MORE_ROWS</code> . | <code>-1</code> | An error has occurred. | <code>-2</code> | <code>dbtextsize</code> has been called for RPC data. |
| <code>dbtextsize</code> returns | To indicate  |                                 |             |                      |  |                 |                        |                 |   |
| <code>&gt;= 0</code>            | The number of bytes that remain to be read. Zero indicates <code>NO_MORE_ROWS</code> .   |                                 |             |                      |  |                 |                        |                 |   |
| <code>-1</code>                 | An error has occurred.   |                                 |             |                      |  |                 |                        |                 |   |
| <code>-2</code>                 | <code>dbtextsize</code> has been called for RPC data.  |                                 |             |                      |  |                 |                        |                 |   |
| Usage                           | <ul style="list-style-type: none"> <li><code>dbtextsize</code> assumes that there is only one column and that this column is of datatype <code>text</code> or <code>image</code>.</li> <li><code>dbtextsize</code> is useful when an application does not know how large a text or image value is.</li> <li><code>dbtextsize</code> does not work with RPC text data.</li> </ul>   |                                 |             |                      |  |                 |                        |                 |   |
| See also                        | <code>dbreadtext</code>  |                                 |             |                      |  |                 |                        |                 |   |

## dbtsnewlen

|              |   |
|--------------|---|
| Description  | Return the length of the new value of the <i>timestamp</i> column after a browse-mode update.   |
| Syntax       | <pre>int dbtsnewlen(dbproc)</pre> <p><code>DBPROCESS *dbproc;</code></p>  |
| Parameters   | <p><code>dbproc</code></p> <p>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>                              |
| Return value | The length (in bytes) of the updated row's new timestamp value. If no timestamp was returned to the application (possibly because the update was unsuccessful, or because the update statement did not contain the <code>tsequal</code> built-in function), <code>dbtsnewlen</code> will return <code>-1</code> . |

|          |   |
|----------|---|
| Usage    | <ul style="list-style-type: none"><li>• dbtsnewlen is one of the DB-Library browse-mode routines. See “Browse mode” on page 26 for a detailed discussion of browse mode.</li><li>• dbtsnewlen provides information about the <i>timestamp</i> column. The where clause returned by dbqual contains a call to the tsequal built-in function. When such a where clause is used in an update statement, the tsequal function places a new value in the updated row’s <i>timestamp</i> column and returns the new timestamp value to the application (if the update is successful). The dbtsnewlen function allows the application to save the length of the new timestamp value, possibly for use with dbtspout.</li></ul> |
| See also | dbcolbrowse, dbcolsource, dbqual, dbtabbrowse, dbtabcount, dbtabname, dbtabsource, dbtsnewval, dbtspout   |

## dbtsnewval

|              |  |
|--------------|--|
| Description  | Return the new value of the <i>timestamp</i> column after a browse-mode update.  |
| Syntax       | DBBINARY *dbtsnewval(dbproc)<br><br>DBPROCESS *dbproc;   |
| Parameters   | dbproc<br>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.  |
| Return value | A pointer to the updated row’s new timestamp value. If no timestamp was returned to the application (possibly because the update was unsuccessful, or because the update statement did not contain the tsequal built-in function), the pointer will be NULL.   |
| Usage        | <ul style="list-style-type: none"><li>• dbtsnewval is one of the DB-Library browse-mode routines. See “Browse mode” on page 26 for a detailed discussion of browse mode.</li><li>• dbtsnewval provides information about the <i>timestamp</i> column. The where clause returned by dbqual contains a call to the tsequal built-in function. When such a where clause is used in an update statement, the tsequal function places a new value in the updated row’s <i>timestamp</i> column and returns the new timestamp value to the application (if the update is successful). This routine allows the application to save the new timestamp value, possibly for use with dbtspout.</li></ul> |

See also dbtabbrowse, dbtabsource, dbqual, dbtabbrowse, dbtabcount, dbtabname, dbtabsource, dbtsnewlen, dbtspu

## dbtspu

**Description** Put the new value of the *timestamp* column into the given table's current row in the DBPROCESS.

**Syntax** RETCODE dbtspu(dbproc, newts, newtslen, tabnum, tabname)

```
DBPROCESS  *dbproc;
DBBINARY   *newts;
int         newtslen;
int         tabnum;
char        *tabname;
```

**Parameters** **dbproc**  
A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

This must be the DBPROCESS used to perform the original select query.

**newts**  
A pointer to the new timestamp value. It is returned by dbtsnewval.

**newtslen**  
The length of the new timestamp value. It is returned by dbtsnewlen.

**tabnum**  
The number of the updated table. Table numbers start at 1. *tabnum* must refer to a browsable table. Use dbtabbrowse to determine whether a table is browsable.

If this value is -1, the *tabname* parameter will be used to identify the table.

**tabname**  
A pointer to a null-terminated table name. *tabname* must refer to a browsable table. If this pointer is NULL, the *tabnum* parameter will be used to identify the table.

**Return value** SUCCEED or FAIL.

The following situations will cause this routine to return FAIL:

|          |   |
|----------|---|
|          | <ul style="list-style-type: none"><li>• The application tries to update the timestamp of a non-existent row.</li><li>• The application tries to update the timestamp using NULL as the timestamp value (<i>newts</i>).</li><li>• The specified table is non-browsable.</li></ul>  |
| Usage    | <ul style="list-style-type: none"><li>• dbtspout is one of the DB-Library browse-mode routines. See “Browse mode” on page 26 for a detailed discussion of browse mode.</li><li>• dbtspout manipulates the timestamp column. The where clause returned by dbqual contains a call to the tsequal built-in function. When such a where clause is used in an update statement, the tsequal function places a new value in the updated row’s timestamp column and returns the new timestamp value to the application (if the update is successful). If the same row is updated a second time, the update statement’s where clause must use the latest timestamp value.</li></ul> <p>This routine updates the timestamp in the DBPROCESS for the row currently being browsed. Then, if the application needs to update the row a second time, it can call dbqual to formulate a new where clause that uses the new timestamp.</p> |
| See also | dbcolbrowse, dbcolsource, dbqual, dbtabbrowse, dbtabcount, dbtabname, dbtabsource, dbtsnewlen, dbtsnewval   |

## dbtxptr

|             |  |
|-------------|--|
| Description | Return the value of the text pointer for a column in the current row.  |
| Syntax      | <pre>DBBINARY *dbtxptr(dbproc, column)</pre> <pre>DBPROCESS *dbproc;<br/>int column;</pre>   |
| Parameters  | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>column</b></p> <p>The number of the select list column of interest. Column numbers start at 1.</p> |



|              |  |
|--------------|--|
| Return value | A DBBINARY pointer to the text pointer for the column of interest. This pointer may be NULL.   |
| Usage        | <ul style="list-style-type: none"> <li>• Every database column row of type SYBTEXT or SYBIMAGE has an associated text pointer, which uniquely identifies the text or image value. This text pointer is used by the dbwritetext function to update text and image values.</li> <li>• It is important that all the rows of the specified text or image column have valid text pointers. A text or image column row will have a valid text pointer if it contains data. However, if the text or image column row contains a null value, its text pointer will be valid only if the null value was explicitly entered with the update statement.</li> </ul> <p>Assume a table textnull with columns key and x, where x is a text column that permits nulls. The following statement assigns valid text pointers to the text column's rows:</p> <pre>update textnull set x = null</pre> <p>On the other hand, the insert of a null value into a text column does not provide a valid text pointer. This is true for an insert of an explicit null or an insert of an implicit null, such as the following:</p> <pre>insert textnull (key) values (2)</pre> <p>When dealing with a null text or image value, be sure to use update to get a valid text pointer.</p> <ul style="list-style-type: none"> <li>• An application must select a row containing a text or image value before calling dbtxptr to return the associated text pointer. The select causes a copy of the text pointer to be placed in the application's DBPROCESS. The application can then retrieve this text pointer from the DBPROCESS using dbtxptr.</li> </ul> <p>If no select is performed prior to the call to dbtxptr, the call will result in a DB-Library error message.</p> <ul style="list-style-type: none"> <li>• For an example that uses dbtxptr, see the dbwritetext reference page.</li> </ul> |
| See also     | dbtxtimestamp, dbwritetext   |

## dbtxtimestamp

|              |   |
|--------------|---|
| Description  | Return the value of the text timestamp for a column in the current row.   |
| Syntax       | <b>DBBINARY</b> *dbtxtimestamp(dbproc, column)  |
| Parameters   | <div><b>DBPROCESS</b> *dbproc;<br/><b>int</b> column;</div> <div><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</div> <div><b>column</b><br/>The number of the select list column of interest. Column numbers start at 1.</div>  |
| Return value | A DBBINARY pointer to the text timestamp for the column of interest. This pointer may be NULL.  |
| Usage        | <ul style="list-style-type: none"><li>• Every database column of type SYBTEXT or SYBIMAGE has an associated text timestamp, which marks the time of the column's last modification. The text timestamp is useful in conjunction with the dbwritetext function, to ensure that two competing application users do not inadvertently wipe out each other's modifications to the same value in the database. It is returned to the DBPROCESS when a Transact-SQL select is performed on a SYBTEXT or SYBIMAGE column.</li><li>• The length of a non-NULL text timestamp is always DBTXTSLEN (currently defined as 8 bytes).</li><li>• An application must select a row containing a text or image value before calling dbtxtimestamp to return the associated text timestamp. The select causes a copy of the text timestamp to be placed in the application's DBPROCESS. The application can then retrieve this text timestamp from the DBPROCESS using dbtxtimestamp.<br/><br/>If no select is performed prior to the call to dbtxtimestamp, the call will result in a DB-Library error message.</li><li>• For an example that uses dbtxtimestamp, see the dbwritetext reference page.</li></ul> |
| See also     | dbtxptr, dbwritetext  |

## dbtxtsnewval

|              |  |
|--------------|--|
| Description  | Return the new value of a text timestamp after a call to <code>dbwritetext</code> .  |
| Syntax       | <pre>DBBINARY *dbtxtsnewval(dbproc)  DBPROCESS  *dbproc;</pre>   |
| Parameters   | <p><code>dbproc</code></p> <p>A pointer to the <code>DBPROCESS</code> structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p>   |
| Return value | A pointer to the new text timestamp value for the <code>SYBTEXT</code> or <code>SYBIMAGE</code> value modified by a <code>dbwritetext</code> operation. This pointer may be <code>NULL</code> .  |
| Usage        | <ul style="list-style-type: none"> <li>• Every database column of type <code>SYBTEXT</code> or <code>SYBIMAGE</code> has an associated text timestamp, which is updated whenever the column's value is changed. The text timestamp is useful in conjunction with the <code>dbwritetext</code> function to ensure that two competing application users do not inadvertently wipe out each other's modifications to the same value in the database. It is returned to the <code>DBPROCESS</code> when a Transact-SQL <code>select</code> is performed on a <code>SYBTEXT</code> or <code>SYBIMAGE</code> column and may be examined by calling <code>dbtxttimestamp</code>.</li> <li>• After each successful <code>dbwritetext</code> operation (which may include a number of calls to <code>dbmoretext</code>), the server will send the updated text timestamp value back to DB-Library. <code>dbtxtsnewval</code> provides a way for the application to get this new timestamp value.</li> <li>• The application can use <code>dbtxtsnewval</code> in two ways. First, the return from <code>dbtxtsnewval</code> can be used as the <i>timestamp</i> parameter of a <code>dbwritetext</code> call. Second, <code>dbtxtsnewval</code> and <code>dbtxtsput</code> can be used together to put the new timestamp value into the <code>DBPROCESS</code> row buffer, for future access using <code>dbtxttimestamp</code>. This is particularly useful when the application is buffering result rows and does not need the new timestamp immediately.</li> </ul> |
| See also     | <code>dbmoretext</code> , <code>dbtxttimestamp</code> , <code>dbtxtsput</code> , <code>dbwritetext</code>  |

## dbtxtsput

|             |  |
|-------------|--|
| Description | Put the new value of a text timestamp into the specified column of the current row in the <code>DBPROCESS</code> . |
|-------------|--|

|              |   |
|--------------|---|
| Syntax       | <pre>RETCODE dbtxtsput(dbproc, newtxts, colnum)  DBPROCESS *dbproc; DBBINARY  *newtxts; int        colnum;</pre>  |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>newtxts</b><br/>A pointer to the new text timestamp value. It is returned by dbtxtsnewval.</p> <p><b>colnum</b><br/>The number of the select list column of interest. Column numbers start at 1.</p>   |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"><li>• Every database column of type SYBTEXT or SYBIMAGE has an associated text timestamp, which is updated whenever the column's value is changed. The text timestamp is useful in conjunction with the dbwritetext function, to ensure that two competing application users do not inadvertently wipe out each other's modifications to the same value in the database. It is returned to the DBPROCESS when a Transact-SQL select is performed on a SYBTEXT or SYBIMAGE column and may be examined by calling dbtxtimestamp.</li><li>• After each successful dbwritetext operation (which may include a number of calls to dbmoretext), the server will send the updated text timestamp value back to DB-Library. dbtxtsnewval allows the application to get this new timestamp value. The application can then use dbtxtsput to put the new timestamp value into the DBPROCESS row buffer, for future access using dbtxtimestamp. This is particularly useful when the application is buffering result rows and does not need the new timestamp immediately.</li></ul> |
| See also     | dbmoretext, dbtxtimestamp, dbtxtsnewval, dbwritetext  |

## dbuse

|             |  |
|-------------|--|
| Description | Use a particular database.               |
| Syntax      | <pre>RETCODE dbuse(dbproc, dbname)</pre> |

|              |  |
|--------------|--|
|              | <pre>DBPROCESS *dbproc; char      *dbname;</pre>   |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>dbname</b><br/>The name of the database to use.</p>   |
| Return value | SUCCEED or FAIL.   |
| Usage        | <ul style="list-style-type: none"> <li>• This routine issues a Transact-SQL use command for the specified database for a particular DBPROCESS. It sets up the command and calls <code>dbsqlxexec</code> and <code>dbresults</code>.</li> <li>• If the use command fails because the requested database has not yet completed a recovery process, <code>dbuse</code> will continue to send use commands at one second intervals until it either succeeds or encounters some other error.</li> <li>• The routine uses the <i>dbproc</i> provided by the caller. It also uses the command buffer of that <code>dbproc</code>. <code>dbuse</code> overwrites any existing commands in the buffer and clears the buffer when it is finished.</li> </ul> |
| See also     | <code>dbchange</code> , <code>dbname</code>  |

## dbvarylen

|             |  |
|-------------|--|
| Description | Determine whether the specified regular result column's data can vary in length.   |
| Syntax      | <pre>DBBOOL dbvarylen(dbproc, column)</pre> <pre>DBPROCESS *dbproc; int        column;</pre>   |
| Parameters  | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> |

|              |   |
|--------------|---|
|              | <p>column</p> <p>The number of the regular result column of interest. The first column is number 1.</p>   |
| Return value | <p>“TRUE” or “FALSE”, indicating whether or not the column’s data can vary in length. dbvarylen also returns “FALSE” if the column number is out of range.</p>  |
| Usage        | <ul style="list-style-type: none"><li>• This routine indicates whether a particular regular (that is, non-compute) result column’s data can vary in length. It will return “TRUE” if the result column is derived from a database column of type varchar, varbinary, text, image, boundary, or sensitivity. It will also return “TRUE” if the source database column is defined as NULL, meaning that it may contain a null value.</li><li>• This routine is useful with programs that handle ad hoc queries, if the program needs to be alerted to the possibility of null or variable length data.</li><li>• You can use dbcoltype to determine a column’s datatype. See Types on page 412 for a list of datatypes.</li></ul> |
| See also     | <p>dbcollen, dbcolname, dbcoltype, dbdata, dbdatlen, dbnumcols, dbprtype</p>  |

## dbversion

|              |  |
|--------------|--|
| Description  | <p>Determine which version of DB-Library is in use.</p>  |
| Syntax       | <p>char *dbversion()</p>   |
| Parameters   | <p>None.</p>   |
| Return value | <p>A pointer to a character string containing the version of DB-Library in use.</p>  |
| Usage        | <p>dbversion returns a pointer to a character string that contains the version number for the DB-Library that is currently in use.</p> |
| See also     | <p>DBTDS</p>   |

## dbwillconvert

|              |   |
|--------------|---|
| Description  | Determine whether a specific datatype conversion is available within DB-Library.  |
| Syntax       | <pre>DBBOOL dbwillconvert(srctype, desttype)</pre><br><pre>int    srctype;<br/>int    desttype;</pre>   |
| Parameters   | <p><b>srctype</b><br/>The datatype of the data that is to be converted. This parameter can be any of the server datatypes, as listed in Table 2-30.</p> <p><b>desttype</b><br/>The datatype that the source data is to be converted into. This parameter can be any of the server datatypes, as listed in Table 2-30.</p>   |
| Return value | “TRUE” if the datatype conversion is supported, “FALSE” if the conversion is not supported.   |
| Usage        | <ul style="list-style-type: none"><li>• This routine allows the program to determine whether dbconvert is capable of performing a specific datatype conversion. When dbconvert is asked to perform a conversion that it does not support, it calls a user-supplied error handler (if any), sets a global error number, and returns FAIL.</li><li>• dbconvert can convert data stored in any of the server datatypes (although, of course, not all conversions are legal). Table 2-30 lists the Server and DB-Library datatypes.</li></ul> |

**Table 2-30: Server and DB-Library datatypes**

| <b>Server type</b> | <b>Program variable type</b> |
|--------------------|------------------------------|
| SYBCHAR            | DBCHAR                       |
| SYBTEXT            | DBCHAR                       |
| SYBBINARY          | DBBINARY                     |
| SYBIMAGE           | DBBINARY                     |
| SYBINT1            | DBTINYINT                    |
| SYBINT2            | DBSMALLINT                   |
| SYBINT4            | DBINT                        |
| SYBFLT8            | DBFLT8                       |
| SYBREAL            | DBREAL                       |
| SYBNUMERIC         | DBNUMERIC                    |
| SYBDECIMAL         | DBDECIMAL                    |
| SYBBIT             | DBBIT                        |
| SYBMONEY           | DBMONEY                      |
| SYBMONEY4          | DBMONEY4                     |
| SYBDATETIME        | DBDATETIME                   |
| SYBDATETIME4       | DBDATETIME4                  |
| SYBBOUNDARY        | DBCHAR                       |
| SYBSENSITIVITY     | DBCHAR                       |

- Table 2-8 on page 111 lists the datatype conversions that dbconvert and dbconvert\_ps support. The source datatypes are listed down the leftmost column and the destination datatypes are listed along the top row of the table. (For brevity, the prefix “SYB” has been eliminated from each datatype.) If dbwillconvert returns “TRUE” (T), the conversion is supported; if it returns “FALSE” (F), the conversion is not supported.
- See the reference pages for dbconvert or dbconvert\_ps.

See also

dbaltbind, dbbind, dbconvert, dbconvert\_ps, Types on page 412



## dbwritepage

### Description

Write a page of binary data to the server.

---

**Warning!** Use this routine only if you are absolutely sure you know what you are doing!

---

### Syntax

RETCODE dbwritepage(dbproc, dbname, pageno, size, buf)

```
DBPROCESS  *dbproc;  
char        *dbname;  
DBINT       pageno;  
DBINT       size;  
BYTE        buf[];
```

### Parameters

dbproc

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

dbname

The name of the database of interest.

pageno

The number of the database page to be written.

size

The number of bytes to be written to the server. Currently, Adaptive Server Enterprise database pages are 2048 bytes long.

buf

A pointer to a buffer that holds the data to be written.

### Return value

SUCCEED or FAIL.

### Usage

dbwritepage writes a page of binary data to the server. This routine is useful primarily for examining and repairing damaged database pages. After calling dbwritepage, the DBPROCESS may contain some error or informational messages from the server. These messages may be accessed through a user-supplied message handler.

### See also

dbmsghandle, dbreadpage

## dbwritetext

|             |   |
|-------------|---|
| Description | Send a text or image value to the server.   |
| Syntax      | <pre>RETCODE dbwritetext(dbproc, objname, textptr,<br/>                    textptrlen, timestamp, log,<br/>                    size, text)</pre><br><pre>DBPROCESS  *dbproc;<br/>char        *objname;<br/>DBBINARY    *textptr;<br/>DBTINYINT   textptrlen;<br/>DBBINARY    *timestamp;<br/>DBBOOL      log;<br/>DBINT       size;<br/>BYTE        *text;</pre>  |
| Parameters  | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.</p> <p><b>objname</b><br/>The database table and column name that is separated by a period.</p> <p><b>textptr</b><br/>A pointer to the text pointer of the text or image value to be modified. This can be obtained by calling dbtxptr. The text pointer must be a valid one, as described on the dbtxptr reference page.</p> <p><b>textptrlen</b><br/>This parameter is included for future compatibility. For now, its value must be the defined constant DBTXPLEN.</p> <p><b>timestamp</b><br/>A pointer to the text timestamp of the text or image value to be modified. This can be obtained using dbtxtimestamp or dbtxtsnewval. This value changes whenever the text or image value itself is changed. This parameter is optional and may be passed as NULL.</p> <p><b>log</b><br/>A boolean value specifying whether this dbwritetext operation should be recorded in the transaction log.</p> <p><b>size</b><br/>The total size, in bytes, of the text or image value to be written. Since dbwritetext uses this parameter as its only guide to determining how many bytes to send, <i>size</i> must not exceed the actual size of the value.</p> |

text

The address of a buffer containing the text or image value to be written. If this pointer is NULL, the application must subsequently call `dbmoretext` one or more times, until all *size* bytes of data have been sent to the server.

Return value

SUCCEED or FAIL.

A common cause for failure is an invalid *timestamp* parameter. This occurs if, between the time the application retrieves the text column and the time the application calls `dbwritetext` to update it, a second application intervenes with its own update.

Usage

- `dbwritetext` updates SYBTEXT and SYBIMAGE values. It allows the application to send long values to the server without having to copy them into a Transact-SQL update statement. In addition, `dbwritetext` gives applications access to the text timestamp mechanism, which can be used to ensure that two competing application users do not inadvertently wipe out each other's modifications to the same value in the database.
- The *timestamp* parameter is optional.

If the *timestamp* parameter is supplied, `dbwritetext` succeeds only if the value of the *timestamp* parameter matches the text column's timestamp in the database. If a match occurs, `dbwritetext` updates the text column and at the same time updates the column's timestamp with the current time. This has the effect of governing updates by competing applications—an application's `dbwritetext` call fails if a second application updated the text column between the time the first application retrieved the column and the time it made its `dbwritetext` call.

If the *timestamp* parameter is not supplied, `dbwritetext` updates the text column regardless of the value of the column's timestamp.

- The value to use as the *timestamp* parameter is placed in an application's DBPROCESS when the application performs a select on a text or image value. It can be retrieved from the DBPROCESS using `dbtxttimestamp`.

In addition, after each successful `dbwritetext` operation, which may include a number of calls to `dbmoretext`, Adaptive Server Enterprise sends a new text timestamp value back to DB-Library. `dbtxtsnewval` provides a way for an application to retrieve this new value.

- dbwritetext is similar in function to the Transact-SQL writetext command. It is usually more efficient to call dbwritetext than to send a writetext command through the command buffer. In addition, dbwritetext can handle columns up to 2GB in length, while writetext data is limited to approximately 120K. See the *Adaptive Server Enterprise Reference Manual*.
- dbwritetext can be invoked with or without logging, according to the value of the *log* parameter.

While logging aids media recovery, logging text data quickly increases the size of the transaction log. If you are logging dbwritetext operations, make sure that the transaction log resides on a separate database device. For details, see the *Adaptive Server Enterprise System Administration Guide*, the *create database* reference page, and the *sp\_logdevice* reference page in the *Adaptive Server Enterprise Reference Manual* for details.

To use dbwritetext with logging turned off, the database option *select into/bulkcopy* must be set to “true”. The following SQL command will do this:

```
sp_dboption 'mydb', 'select into/bulkcopy', 'true'
```

See the *Adaptive Server Enterprise Reference Manual* for further details on *sp\_dboption*.

- The application can send a text or image value to the server all at once or a chunk at a time. dbwritetext by itself handles sending an entire text or image value. The use of dbwritetext with dbmoretext allows the application to send a large text or image value to the server in the form of a number of smaller chunks. This is particularly useful with operating systems unable to allocate extremely long data buffers.
- Sending an entire text or image value requires a non-NULL *text* parameter. Then, dbwritetext will execute the data transfer from start to finish, including any necessary calls to *dbsqlok* and *dbresults*. Here is a code fragment that illustrates this use of dbwritetext:

```
LOGINREC      *login;
DBPROCESS     *q_dbproc;
DBPROCESS     *u_dbproc;
DBCHAR        abstract_var[512];

/* Initialize DB-Library. */
if (dbinit() == FAIL)
    exit (ERREXIT);
/*
```

```

** Open separate DBPROCESSes for querying and updating.
** This is not strictly necessary in this example,
** which retrieves only one row. However, this
** approach becomes essential when performing updates
** on multiple rows of retrieved data.
*/
login = dblogin();
q_dbproc = dbopen(login, NULL);
u_dbproc = dbopen(login, NULL);

/* The database column "abstract" is a text column.
** Retrieve the value of one of its rows.
*/
dbcmd(q_dbproc, "select abstract from articles where \
    article_id = 10");
dbsqlxexec(q_dbproc);
dbresults(q_dbproc);
dbbind(q_dbproc, 1, STRINGBIND, (DBINT) 0,
    abstract_var);

/*
** For simplicity, we'll assume that just one row is
** returned.
*/
dbnextrow(q_dbproc);

/* Here we can change the value of "abstract_var" */
/* For instance ... */
strcpy(abstract_var, "A brand new value.");

/* Update the text column */
dbwritetext (u_dbproc, "articles.abstract",
    dbtxptr(q_dbproc, 1), DBTXPLEN,
    dbtxtimestamp(q_dbproc, 1), TRUE,
    (DBINT)strlen(abstract_var), abstract_var);
/* We're all done */
dbexit();

```

- To send chunks of text or image, rather than the whole value at once, set the *text* parameter to NULL. Then, `dbwritetext` will return control to the application immediately after notifying the server that a text transfer is about to begin. The actual text will be sent to the server with `dbmoretext`, which can be called multiple times, once for each chunk. Here is a code fragment that illustrates the use of `dbwritetext` with `dbmoretext`:

```

LOGINREC      *login;
DBPROCESS     *q_dbproc;

```

```
DBPROCESS      *u_dbproc;
DBCHAR         part1[512];
static DBCHAR   part2[512] = " This adds another \
        sentence to the text.";

if (dbinit() == FAIL)
    exit(ERREXIT);

login = dblogin();
q_dbproc = dbopen(login, NULL);
u_dbproc = dbopen(login, NULL);

dbcmd(q_dbproc, "select abstract from articles where \
        article_id = 10");
dbsqlxec(q_dbproc);
dbresults(q_dbproc);
dbbind(q_dbproc, 1, STRINGBIND, (DBINT) 0, part1);

/*
** For simplicity, we'll assume that just one row is
** returned.
*/
dbnextrow(q_dbproc);

/*
** Here we can change the value of part of the text
** column. In this example, we will merely add a
** sentence to the end of the existing text.
*/

/* Update the text column */
dbwritetext (u_dbproc, "articles.abstract",
dbtxptr(q_dbproc, 1), DBTXPLEN,
dbtxtimestamp(q_dbproc, 1), TRUE,
(DBINT)(strlen(part1) + strlen(part2)), NULL);

dbsqllok(u_dbproc);
dbresults(u_dbproc);

/* Send the update value in chunks */
dbmoretext(u_dbproc, (DBINT)strlen(part1), part1);
dbmoretext(u_dbproc, (DBINT)strlen(part2), part2);

dbsqllok(u_dbproc);
dbresults(u_dbproc);
dbexit();
```

Note the required calls to `dbsqllok` and `dbresults` between the call to `dbwritetext` and the first call to `dbmoretext`, and after the final call to `dbmoretext`.

- When `dbwritetext` is used with `dbmoretext`, it locks the specified database text column. The lock is not released until the final `dbmoretext` has sent its data. This ensures that a second application does not read or update the text column in the midst of the first application's update.
- You cannot use `dbwritetext` on text or image columns in views.
- The DB-Library/C option `DBTEXTSIZE` affects the value of the server `@@textsize` global variable, which restricts the size of text or image values that Adaptive Server Enterprise returns. `@@textsize` has a default value of 32,768 bytes. An application that retrieves text or image values larger than 32,768 bytes will need to call `dbsetopt` to make `@@textsize` larger.
- The DB-Library/C option `DBTEXTLIMIT` limits the size of text or image values that DB-Library/C will read.

See also

`dbmoretext`, `dbtxptr`, `dbtxtimestamp`, `dbwritetext`, `dbtxtsput`

## dbxlate

Description

Translate a character string from one character set to another.

Syntax

```
int dbxlate(dbproc, src, srclen, dest, destlen, xlt,
            srcbytes_used, srcend, status)
```

```
DBPROCESS dbproc;
char      *src;
int       srclen;
char      *dest;
int       destlen;
DBXLATE   *xlt;
int       *srcbytes_used;
DBBOOL    srcend;
int       *status;
```

Parameters

`dbproc`

A pointer to the `DBPROCESS` structure that provides the connection for a particular front-end/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.

`src`

A pointer to the string to be translated.

- srclen
- The length, in bytes, of *src*. If *srclen* is -1, *src* is assumed to be null-terminated.
- dest
- A pointer to the buffer to contain the translated string, including a null terminator.
- destlen
- The size, in bytes, of the buffer to contain the translated string. If *destlen* is -1, *dest* is assumed to be large enough to hold the translated string and its null terminator.
- xlt
- A pointer to a translation structure used to translate character strings from one character set to another. The translation structure is allocated using `dbload_xlate`.
- srcbytes\_used
- The number of bytes actually translated. If the fully translated string would overflow *dest*, `dbxlate` translates only as much of *src* as will fit. If *destlen* is -1, *srcbytes\_used* is *srclen*.
- srcend
- A boolean value indicating whether or not more data is arriving. If *srcend* is “true”, no more data is arriving. If *srcend* is “false”, *src* is part of a larger string of data to be translated, and it is not the end of the string.
- status
- A pointer to a code indicating the status of the translated character string. Table 2-31 lists the possible values for *status*.

**Table 2-31: Values for status**

| Value of status | To indicate  |
|-----------------|--|
| DBXLATE_XOF     | The translated string overflowed <i>dest</i> .   |
| DBXLATE_XOK     | The translation succeeded.   |
| DBXLATE_XPAT    | The last bytes of <i>src</i> are the beginning of a pattern for which there is a translation. These bytes were not translated. |

- Return value
- The number of bytes actually placed in *dest* on success; a negative integer on error.
- Usage
- `dbxlate` translates a character string from one character set to another. It is useful when the server character set differs from the display device’s character set.
  - The following code fragment illustrates the use of `dbxlate`:



```

char          destbuf[128];
int           srcbytes_used;
DBXLATE       *xlt_todisp;
DBXLATE       *xlt_tosrv;

dbload_xlate((DBPROCESS *)NULL, "iso_1",
             "trans.xlt", &xlt_tosrv, &xlt_todisp);
printf("Original string: \n\t%s\n\n",
       TEST_STRING);
dbxlate((DBPROCESS *)NULL, TEST_STRING,
        strlen(TEST_STRING), destbuf, -1, xlt_todisp,
        &srcbytes_used);
printf("Translated to display character set: \
\t\t%s\n\n", destbuf);
dbfree_xlate((DBPROCESS *)NULL, xlt_tosrv,
             xlt_todisp);

```

See also `dbload_xlate`, `dbfree_xlate`

## Errors

|             |   |
|-------------|---|
| Description | The complete collection of DB-Library errors and error severities.  |
| Syntax      | <pre>#include &lt;sybfront.h&gt;  #include &lt;sybdb.h&gt;  #include &lt;syberror.h&gt;</pre>   |
| Usage       | <ul style="list-style-type: none"> <li>• This is the complete list of possible DB-Library errors and error severities.</li> <li>• The error values are listed alphabetically in Table 2-32 on page 391. The second column of this table gives the error severity for each error as a symbolic value. The third column contains the text associated with the error.</li> <li>• Table 2-33 on page 406 provides a list of all possible error severities, with their numerical equivalents and an explanation of the type of error.</li> <li>• When an error or informational event occurs, these numbers are passed to the application's current error handler (if any). An application calls <code>dberrhandle</code> to install an error handler.</li> <li>• Error values are defined in the header file <code>sybdb.h</code>. Error severity values are defined in the header file <code>syberror.h</code>. Your program needs to include <code>syberror.h</code> only if it refers to the symbolic error severities.</li> </ul> |

### Errors

Table 2-32 lists all the DB-Library errors.

**Table 2-32: Errors**

| Error name | Error severity | Error text  |
|------------|----------------|---|
| SYBEAAMT   | EXPROGRAM      | User attempted a dbaltbind with mismatched column and variable types.                             |
| SYBEABMT   | EXPROGRAM      | User attempted a dbbind with mismatched column and variable types.                                |
| SYBEABNC   | EXPROGRAM      | Attempt to bind to a non-existent column.   |
| SYBEABNP   | EXPROGRAM      | Attempt to bind using NULL pointers.  |
| SYBEABNV   | EXPROGRAM      | Attempt to bind to a NULL program variable.   |
| SYBEACNV   | EXCONVERSION   | Attempt to do data-conversion with NULL destination variable.                                     |
| SYBEADST   | EXCONSISTENCY  | International Release: Error in attempting to determine the size of a pair of translation tables. |
| SYBEAICF   | EXCONSISTENCY  | International Release: Error in attempting to install custom format.                              |
| SYBEALTT   | EXCONSISTENCY  | International Release: Error in attempting to load a pair of translation tables.                  |
| SYBEAOLF   | EXRESOURCE     | International Release: Error in attempting to open a localization file.                           |
| SYBEAPCT   | EXCONSISTENCY  | International Release: Error in attempting to perform a character set translation.                |
| SYBEAPUT   | EXPROGRAM      | Attempt to print unknown token.   |
| SYBEARDI   | EXRESOURCE     | International Release: Error in attempting to read datetime information from a localization file. |
| SYBEARDL   | EXRESOURCE     | International Release: Error in attempting to read the <i>dblib.loc</i> localization file.        |
| SYBEASEC   | EXPROGRAM      | Attempt to send an empty command buffer to the server.  |

| Error name   | Error severity | Error text   |
|--------------|----------------|--|
| SYBEASNL     | EXPROGRAM      | Attempt to set fields in a null LOGINREC.  |
| SYBEASTL     | EXPROGRAM      | Synchronous I/O attempted at AST level.  |
| SYBEASUL     | EXPROGRAM      | Attempt to set unknown LOGINREC field.   |
| SYBEAUTN     | EXPROGRAM      | Attempt to update the timestamp of a table that has no timestamp column.   |
| SYBEBADPK    | EXINFO         | Packet size of %1 not supported-size of %2 used instead!   |
| SYBEBBCI     | EXINFO         | Batch successfully bulk copied to the server.  |
| SYBEBBL      | EXPROGRAM      | Bad <i>bindlen</i> parameter passed to <i>dbsetnull</i> .  |
| SYBEBCBC     | EXPROGRAM      | <i>bcp_columns</i> must be called before <i>bcp_colfmt</i> and <i>bcp_colfmt_ps</i> .  |
| SYBEBCBNPR   | EXPROGRAM      | <i>bcp_bind</i> : if <i>varaddr</i> is NULL, <i>prefixlen</i> must be 0 and no terminator should be specified.   |
| SYBEBCBNTYP  | EXPROGRAM      | <i>bcp_bind</i> : if <i>varaddr</i> is NULL and <i>varlen</i> greater than 0, the table column type must be SYBTEXT or SYBIMAGE and the program variable type must be SYBTEXT, SYBCHAR, SYBIMAGE or SYBBINARY. |
| SYBEBCBPREF  | EXPROGRAM      | Illegal prefix length. Legal values are 0, 1, 2 or 4.  |
| SYBEBCFO     | EXUSER         | <i>bcp</i> host files must contain at least one column.  |
| SYBEBCHLEN   | EXPROGRAM      | <i>host_colln</i> should be greater than or equal to -1.   |
| SYBEBCIS     | EXCONSISTENCY  | Attempt to bulk copy an illegally-sized column value to the server.  |
| SYBEBCIT     | EXPROGRAM      | It is illegal to use BCP terminators with program variables other than SYBCHAR, SYBBINARY, SYBTEXT, or SYBIMAGE.   |
| SYBEBCITBLEN | EXPROGRAM      | <i>bcp_init: tblname</i> parameter is too long.  |

| Error name  | Error severity | Error text   |
|-------------|----------------|--|
| SYBEBCTBNM  | EXPROGRAM      | bcp_init: <i>tblname</i> parameter cannot be NULL.   |
| SYBEBCMTXT  | EXPROGRAM      | bcp_moretext may be used only when there is at least one text or image column in the Server table.   |
| SYBEBCNL    | EXNONFATAL     | Negative length-prefix found in BCP datafile.  |
| SYBEBCNN    | EXUSER         | Attempt to bulk copy a NULL value into a Server column which does not accept null values.  |
| SYBEBCNT    | EXUSER         | Attempt to use Bulk Copy with a non-existent Server table.   |
| SYBEBCOR    | EXCONSISTENCY  | Attempt to bulk copy an oversized row to the server.   |
| SYBEBCPB    | EXPROGRAM      | bcp_bind, bcp_moretext and bcp_sendrow may not be used after bcp_init has been passed a non-NULL input file name.                                    |
| SYBEBCPCTYP | EXPROGRAM      | bcp_colfmt: If <i>table_colnum</i> is 0, <i>host_type</i> cannot be 0.   |
| SYBEBCPI    | EXPROGRAM      | bcp_init must be called before any other bcp routines.   |
| SYBEBCPN    | EXPROGRAM      | bcp_bind, bcp_collen, bcp_colptr, bcp_moretext and bcp_sendrow may be used only after bcp_init has been called with the copy direction set to DB_IN. |
| SYBEBCPREC  | EXNONFATAL     | Column %1!: Illegal precision value encountered.   |
| SYBEBCPREF  | EXPROGRAM      | Illegal prefix length. Legal values are -1, 0, 1, 2 or 4.  |
| SYBEBCRE    | EXNONFATAL     | I/O error while reading bcp datafile.  |
| SYBEBCRO    | EXINFO         | The BCP hostfile '%1!' contains only %2! rows. It was impossible to read the requested %3! rows.   |
| SYBEBCSA    | EXUSER         | The BCP hostfile '%1!' contains only %2! rows. Skipping all of these rows is not allowed.  |
| SYBEBCSET   | EXCONSISTENCY  | Unknown character set encountered.   |

| Error name   | Error severity | Error text  |
|--------------|----------------|---|
| SYBEBCSI     | EXPROGRAM      | Host-file columns may be skipped only when copying into the Server.   |
| SYBEBCSNDROW | EXPROGRAM      | bcp_sendrow may not be called unless all text data for the previous row has been sent using bcp_moretext.   |
| SYBEBCSNTYP  | EXPROGRAM      | column number %1!: If <i>varaddr</i> is NULL and <i>varlen</i> greater than 0, the table column type must be SYBTEXT or SYBIMAGE and the program variable type must be SYBTEXT, SYBCHAR, SYBIMAGE or SYBBINARY. |
| SYBEBCUC     | EXRESOURCE     | bcp: Unable to close host datafile.   |
| SYBEBCUO     | EXRESOURCE     | bcp: Unable to open host datafile.  |
| SYBEBCVH     | EXPROGRAM      | bcp_exec may be called only after bcp_init has been passed a valid host file.   |
| SYBEBCVLEN   | EXPROGRAM      | <i>varlen</i> should be greater than or equal to -1.  |
| SYBEBCWE     | EXNONFATAL     | I/O error while writing bcp datafile.   |
| SYBEBDIO     | EXPROGRAM      | Bad bulk copy direction. Must be either IN or OUT.  |
| SYBEBEOF     | EXNONFATAL     | Unexpected EOF encountered in bcp datafile.   |
| SYBEBIHC     | EXPROGRAM      | Incorrect host-column number found in bcp format file.  |
| SYBEBIVI     | EXPROGRAM      | bcp_columns, bcp_colfmt and bcp_colfmt_ps may be used only after bcp_init has been passed a valid input file.   |
| SYBEBNCR     | EXPROGRAM      | Attempt to bind user variable to a non-existent compute row.  |
| SYBEBNUM     | EXPROGRAM      | Bad <i>numbytes</i> parameter passed to dbstrcpy.   |
| SYBEBPKS     | EXPROGRAM      | In DBSETLPACKET, the packet size parameter must be between 0 and 999999.  |
| SYBEBPREC    | EXPROGRAM      | Illegal precision specified.  |

| Error name     | Error severity | Error text   |
|----------------|----------------|--|
| SYBEBPROBADDEF | EXCONSISTENCY  | bcp protocol error: Illegal default column ID received.  |
| SYBEBPROCOL    | EXCONSISTENCY  | bcp protocol error: Returned column count differs from the actual number of columns received.        |
| SYBEBPRODEF    | EXCONSISTENCY  | bcp protocol error: Expected default information and got none.                                       |
| SYBEBPRODEFID  | EXCONSISTENCY  | bcp protocol error: Default column ID and actual column ID are not same.                             |
| SYBEBPRODEFTYP | EXCONSISTENCY  | bcp protocol error: Default value datatype differs from column datatype.                             |
| SYBEBPROEXTDEF | EXCONSISTENCY  | bcp protocol error: More than one row of default information received.                               |
| SYBEBPROEXTRES | EXCONSISTENCY  | bcp protocol error: Unexpected set of results received.  |
| SYBEBPRONODEF  | EXCONSISTENCY  | bcp protocol error: Default value received for column that does not have default.                    |
| SYBEBPRONUMDEF | EXCONSISTENCY  | bcp protocol error: Expected number of defaults differs from the actual number of defaults received. |
| SYBEBRFF       | EXRESOURCE     | I/O error while reading bcp format file.   |
| SYBEBSCALE     | EXPROGRAM      | Illegal scale specified.   |
| SYBEBTMT       | EXPROGRAM      | Attempt to send too much text data using the bcp_moretext call.                                      |
| SYBEBTOK       | EXCOMM         | Bad token from the server: Datastream processing out of sync.  |
| SYBEBTYP       | EXPROGRAM      | Unknown bind type passed to DB-Library function.   |
| SYBEBTYPDRV    | EXPROGRAM      | Datatype is not supported by the server.   |
| SYBEBUCE       | EXRESOURCE     | bcp: Unable to close error file.   |
| SYBEBUCF       | EXPROGRAM      | bcp: Unable to close format file.  |
| SYBEBUDF       | EXPROGRAM      | bcp: Unrecognized datatype found in format file.   |
| SYBEBUFF       | EXPROGRAM      | bcp: Unable to create format file.   |

| Error name    | Error severity | Error text   |
|---------------|----------------|--|
| SYBEBUFL      | EXCONSISTENCY  | DB-Library internal error-send buffer length corrupted.  |
| SYBEBUOE      | EXRESOURCE     | bcp: Unable to open error file.  |
| SYBEBUOF      | EXPROGRAM      | bcp: Unable to open format file.   |
| SYBEBWEF      | EXNONFATAL     | I/O error while writing bcp error file.  |
| SYBEBWFF      | EXRESOURCE     | I/O error while writing bcp format file.   |
| SYBECAP       | EXCOMM         | DB-Library capabilities not accepted by the Server.  |
| SYBECAPTYP    | EXCOMM         | Unexpected capability type in CAPABILITY datastream.   |
| SYBECDNS      | EXCONSISTENCY  | Datastream indicates that a compute column is derived from a non-existent select list member.                          |
| SYBECDOMAIN   | EXCONVERSION   | Source field value is not within the domain of legal values.   |
| SYBECINTERNAL | EXCONVERSION   | Internal Conversion error.   |
| SYBECLOS      | EXCOMM         | Error in closing network connection.   |
| SYBECLPR      | EXCONVERSION   | Data conversion resulted in loss of precision.   |
| SYBECNOR      | EXPROGRAM      | Column number out of range.  |
| SYBECNOV      | EXCONVERSION   | Attempt to set variable to NULL resulted in overflow.  |
| SYBECOFL      | EXCONVERSION   | Data conversion resulted in overflow.  |
| SYBECONN      | EXCOMM         | Unable to connect: Adaptive Server Enterprise is unavailable or does not exist.  |
| SYBECRNC      | EXPROGRAM      | The current row is not a result of compute clause %1!, so it is illegal to attempt to extract that data from this row. |
| SYBECRSAGR    | EXPROGRAM      | Aggregate functions are not allowed in a cursor statement.   |
| SYBECRSBROL   | EXPROGRAM      | Backward scrolling cannot be used in a forward scrolling cursor.   |



| Error name     | Error severity | Error text  |
|----------------|----------------|---|
| SYBECRSBSKEY   | EXPROGRAM      | Keyset cannot be scrolled backward in mixed cursors with a previous fetch type.                     |
| SYBECRSBUFR    | EXPROGRAM      | Row buffering should not be turned on when using cursor APIs.                                       |
| SYBECRSDIS     | EXPROGRAM      | Cursor statement contains one of the disallowed phrases compute, union, for browse, or select into. |
| SYBECRSFLAST   | EXPROGRAM      | Fetch type LAST requires fully keyset driven cursors.   |
| SYBECRSFRAND   | EXPROGRAM      | Fetch types RANDOM and RELATIVE can only be used within the keyset of keyset driven cursors.        |
| SYBECRSFROWN   | EXPROGRAM      | Row number to be fetched is outside valid range.  |
| SYBECRSFTYPE   | EXRESOURCE     | Unknown fetch type.   |
| SYBECRSINV     | EXPROGRAM      | Invalid cursor statement.   |
| SYBECRSINVALID | EXRESOURCE     | The cursor handle is invalid.   |
| SYBECRSMROWS   | EXRESOURCE     | Multiple rows are returned, only one is expected while retrieving dbname.                           |
| SYBECRSNOBIND  | EXPROGRAM      | Cursor bind must be called prior to dbcursor invocation.  |
| SYBECRSNOCOUNT | EXPROGRAM      | The DBNOCOUNT option should not be turned on when doing updates or deletes with dbcursor.           |
| SYBECRSNOFREE  | EXPROGRAM      | The DBNOAUTOFREE option should not be turned on when using cursor APIs.                             |
| SYBECRSNOIND   | EXPROGRAM      | One of the tables involved in the cursor statement does not have a unique index.                    |
| SYBECRSNOKEYS  | EXRESOURCE     | The entire keyset must be defined for KEYSET type cursors.  |
| SYBECRSNOLEN   | EXRESOURCE     | No unique index found.  |
| SYBECRSNOPTCC  | EXRESOURCE     | No OPTCC was found.   |
| SYBECRSNORDER  | EXRESOURCE     | The order of clauses must be from, where, and order by.   |
| SYBECRSNORES   | EXPROGRAM      | Cursor statement generated no results.  |

| Error name     | Error severity | Error text  |
|----------------|----------------|---|
| SYBECRSNROWS   | EXRESOURCE     | No rows returned, at least one is expected.   |
| SYBECRSNOTABLE | EXRESOURCE     | Table name is NULL.   |
| SYBECRSNOUPD   | EXPROGRAM      | Update or delete operation did not affect any rows.   |
| SYBECRSNOWHERE | EXPROGRAM      | A where clause is not allowed in a cursor update or insert.   |
| SYBECRSNUNIQUE | EXRESOURCE     | No unique keys associated with this view.   |
| SYBECRSORD     | EXPROGRAM      | Only fully keyset driven cursors can have order by, group by, or having phrases.                              |
| SYBECRSRO      | EXPROGRAM      | Data locking or modifications cannot be made in a read-only cursor.   |
| SYBECRSSET     | EXPROGRAM      | A set clause is required for a cursor update or insert.   |
| SYBECRSTAB     | EXPROGRAM      | Table name must be determined in operations involving data locking or modifications.                          |
| SYBECRSVAR     | EXRESOURCE     | There is no valid address associated with this bind.  |
| SYBECRSVIEW    | EXPROGRAM      | A view cannot be joined with another table or a view in a cursor statement.                                   |
| SYBECRSVIIND   | EXPROGRAM      | The view used in the cursor statement does not include all the unique index columns of the underlying tables. |
| SYBECRSUPDNB   | EXPROGRAM      | Update or insert operations cannot use bind variables when binding type is NOBIND.                            |
| SYBECRSUPDTAB  | EXPROGRAM      | Update or insert operations using bind variables require single table cursors.                                |
| SYBECSYN       | EXCONVERSION   | Attempt to convert data stopped by syntax error in source field.  |
| SYBECUFL       | EXCONVERSION   | Data conversion resulted in underflow.  |
| SYBEDBPS       | EXRESOURCE     | Maximum number of DBPROCESSes already allocated.  |

| Error name | Error severity | Error text   |
|------------|----------------|--|
| SYBEDDNE   | EXINFO         | DBPROCESS is dead or not enabled.  |
| SYBEDIVZ   | EXUSER         | Attempt to divide by \$0.00 in function %1!.   |
| SYBEDNTI   | EXPROGRAM      | Attempt to use dbtxtsput to put a new text timestamp into a column whose datatype is neither SYBTEXT nor SYBIMAGE. |
| SYBEDPOR   | EXPROGRAM      | Out-of-range <i>datepart</i> constant.   |
| SYBEDVOR   | EXPROGRAM      | Day values must be between 1 and 7.  |
| SYBEECAN   | EXINFO         | Attempted to cancel unrequested event notification.  |
| SYBEEINI   | EXINFO         | Must call dbreginit before dbregexec.  |
| SYBEETD    | EXPROGRAM      | Failure to send the expected amount of text or image data using dbmoretext.  |
| SYBEEUNR   | EXCOMM         | Unsolicited event notification received.   |
| SYBEEVOP   | EXINFO         | Called dbregwatch with a bad options parameter.  |
| SYBEEVST   | EXINFO         | Must initiate a transaction before calling dbregparam.   |
| SYBEFCON   | EXCOMM         | Adaptive Server Enterprise connection failed.  |
| SYBEFRES   | EXFATAL        | Challenge-Response function failed.  |
| SYBEFSHD   | EXRESOURCE     | Error in attempting to find the Sybase home directory.   |
| SYBEFUNC   | EXPROGRAM      | Functionality not supported at the specified version level.  |
| SYBEICN    | EXPROGRAM      | Invalid <i>computeid</i> or compute column number.   |
| SYBEIDCL   | EXCONSISTENCY  | Illegal datetime column length returned by Adaptive Server Enterprise. Legal datetime lengths are 4 and 8 bytes.   |
| SYBEIDECCL | EXCONSISTENCY  | Invalid decimal column length returned by the server.  |

| Error name   | Error severity | Error text   |
|--------------|----------------|--|
| SYBEIFCL     | EXCONSISTENCY  | Illegal floating-point column length returned by Adaptive Server Enterprise. Legal floating-point lengths are 4 and 8 bytes. |
| SYBEIFNB     | EXPROGRAM      | Illegal field number passed to bcp_control.  |
| SYBEIICL     | EXCONSISTENCY  | Illegal integer column length returned by Adaptive Server Enterprise. Legal integer lengths are 1, 2, and 4 bytes.           |
| SYBEIMCL     | EXCONSISTENCY  | Illegal money column length returned by Adaptive Server Enterprise. Legal money lengths are 4 and 8 bytes.                   |
| SYBEINLN     | EXUSER         | Interface file: unexpected end-of-line.  |
| SYBEINTF     | EXUSER         | Server name not found in interface file.   |
| SYBEINUMCL   | EXCONSISTENCY  | Invalid numeric column length returned by the server.  |
| SYBEIPV      | EXINFO         | %1! is an illegal value for the %2! parameter of %3!.  |
| SYBEISOI     | EXCONSISTENCY  | International Release: Invalid sort-order information found.   |
| SYBEISRVPREC | EXCONSISTENCY  | Illegal precision value returned by the server.  |
| SYBEISRVSCL  | EXCONSISTENCY  | Illegal scale value returned by the server.  |
| SYBEITIM     | EXPROGRAM      | Illegal timeout value specified.   |
| SYBEIVERS    | EXPROGRAM      | Illegal version level specified.   |
| SYBEKBCI     | EXINFO         | 1000 rows sent to the server.  |
| SYBEKBCO     | EXINFO         | 1000 rows successfully bulk copied to host file.   |
| SYBEMEM      | EXRESOURCE     | Unable to allocate sufficient memory.  |
| SYBEMOV      | EXUSER         | Money arithmetic resulted in overflow in function %1!.   |
| SYBEMPLL     | EXUSER         | Attempt to set maximum number of DBPROCESSes lower than 1.   |
| SYBEMVOR     | EXPROGRAM      | Month values must be between 1 and 12.   |

| Error name | Error severity | Error text   |
|------------|----------------|--|
| SYBENBUF   | EXINFO         | Called dbsendpassthru with a NULL <i>buf</i> parameter.                              |
| SYBENBVP   | EXPROGRAM      | Cannot pass dbsetnull a NULL <i>bindval</i> pointer.                                 |
| SYBENDC    | EXPROGRAM      | Cannot have negative component in date in numeric form.                              |
| SYBENDTP   | EXPROGRAM      | Called dbdatecrack with NULL datetime parameter.                                     |
| SYBENEG    | EXCOMM         | Negotiated login attempt failed.   |
| SYBENHAN   | EXINFO         | Called dbrecvpassthru with a NULL handle parameter.                                  |
| SYBENMOB   | EXPROGRAM      | No such member of order by clause.   |
| SYBENOEV   | EXINFO         | DBPOLL can not be called when registered procedure notifications have been disabled. |
| SYBENPRM   | EXPROGRAM      | NULL parameter not allowed for this dboption.  |
| SYBENSIP   | EXPROGRAM      | Negative starting index passed to dbstrcpy.  |
| SYBENTLL   | EXUSER         | Name too long for LOGINREC field.  |
| SYBENTTN   | EXPROGRAM      | Attempt to use dbtxtsput to put a new text timestamp into a non-existent data row.   |
| SYBENULL   | EXINFO         | NULL DBPROCESS pointer passed to DB-Library.   |
| SYBENULP   | EXPROGRAM      | Called %s with a NULL %s parameter.  |
| SYBENXID   | EXNONFATAL     | The Server did not grant us a distributed-transaction ID.                            |
| SYBEONCE   | EXPROGRAM      | Function can be called only once.  |
| SYBEOOB    | EXCOMM         | Error in sending out-of-band data to the server.                                     |
| SYBEOPIN   | EXNONFATAL     | Could not open interface file.   |
| SYBEOPNA   | EXNONFATAL     | Option is not available with current server.   |

| Error name  | Error severity | Error text   |
|-------------|----------------|--|
| SYBEOREN    | EXINFO         | International Release: Warning: an out-of-range error-number was encountered in <i>dblib.loc</i> . The maximum permissible error-number is defined as DBERRCOUNT in <i>sybdb.h</i> . |
| SYBEORPF    | EXUSER         | Attempt to set remote password would overflow the login record's remote password field.  |
| SYBEPOLL    | EXINFO         | There is already an active dbpoll.   |
| SYBERPTF    | EXINFO         | dbtracestring may only be called from a <i>printfunc</i> .   |
| SYBEPWD     | EXUSER         | Login incorrect.   |
| SYBERDCN    | EXCONVERSION   | Requested data conversion does not exist.  |
| SYBERDNR    | EXPROGRAM      | Attempt to retrieve data from a non-existent row.  |
| SYBEREAD    | EXCOMM         | Read from the server failed.   |
| SYBERESP    | EXPROGRAM      | Response function address passed to dbresponse must be non-NULL.   |
| SYBERPCS    | EXINFO         | Must call dbrpcinit before dbrpcparam or dbrpcsend.  |
| SYBERPIL    | EXPROGRAM      | It is illegal to pass -1 to dbrpcparam for the <i>datalen</i> of parameters which are of type SYBCHAR, SYBVARCHAR, SYBBINARY, or SYBVARBINARY.                                       |
| SYBERPNA    | EXNONFATAL     | The RPC facility is available only when using a server whose version number is 4.0 or later.   |
| SYBERPND    | EXPROGRAM      | Attempt to initiate a new Adaptive Server Enterprise operation with results pending.   |
| SYBERPNULL  | EXPROGRAM      | <i>value</i> parameter for dbrpcparam can be NULL, only if the <i>datalen</i> parameter is 0.  |
| SYBERPTXTIM | EXPROGRAM      | RPC parameters cannot be of type text or image.  |

| Error name | Error severity | Error text   |
|------------|----------------|--|
| SYBERPUL   | EXPROGRAM      | When passing a SYBINTN, SYBDATETIMN, SYBMONEYN, or SYBFLTIN parameter using dbrpcparam, it is necessary to specify the parameter's maximum or actual length so that DB-Library can recognize it as a SYINT1, SYBINT2, SYBINT4, SYBMONEY, SYBMONEY4, and so on. |
| SYBERTCC   | EXPROGRAM      | dbreadtext may not be used to receive the results of a query that contains a COMPUTE clause.   |
| SYBERTSC   | EXPROGRAM      | dbreadtext may be used only to receive the results of a query that contains a single result column.  |
| SYBERXID   | EXNONFATAL     | The Server did not recognize our distributed-transaction ID.   |
| SYBESECURE | EXPROGRAM      | Secure Adaptive Server Enterprise function not supported in this version.  |
| SYBESEFA   | EXPROGRAM      | DBSETNOTIFS cannot be called if connections are present.   |
| SYBESEOF   | EXCOMM         | Unexpected EOF from the server.  |
| SYBESFOV   | EXPROGRAM      | International Release: dbsafestr overflowed its destination buffer.  |
| SYBESMSG   | EXSERVER       | General Adaptive Server Enterprise error: Check messages from the server.  |
| SYBESOCK   | EXCOMM         | Unable to open socket.   |
| SYBESPID   | EXPROGRAM      | Called dbspid with a NULL dbproc.  |
| SYBESYNC   | EXCOMM         | Read attempted while out of synchronization with Adaptive Server Enterprise.   |
| SYBETEXS   | EXINFO         | Called dbmoretext with a bad size parameter.   |
| SYBETIME   | EXTIME         | Adaptive Server Enterprise connection timed out.   |
| SYBETMCF   | EXPROGRAM      | Attempt to install too many custom formats using dbfmtinstall.   |

| Error name | Error severity | Error text  |
|------------|----------------|---|
| SYBETMTD   | EXPROGRAM      | Attempt to send too much TEXT data using the dbmoretext call.   |
| SYBETPAR   | EXPROGRAM      | No SYBTEXT or SYBIMAGE parameters were defined.   |
| SYBETPTN   | EXUSER         | Syntax error: Only two periods are permitted in table names.  |
| SYBETRAC   | EXINFO         | Attempted to turn off a trace flag that was not on.   |
| SYBETRAN   | EXINFO         | DBPROCESS is being used for another transaction.  |
| SYBETRAS   | EXINFO         | DB-Library internal error: Trace structure not found.   |
| SYBETRSN   | EXINFO         | Bad <i>numbytes</i> parameter passed to dbtracestring.  |
| SYBETSIT   | EXINFO         | Attempt to call dbtspout with an invalid timestamp.   |
| SYBETTS    | EXUSER         | The table which bulk copy is attempting to copy to a host file is shorter than the number of rows which bulk copy was instructed to skip. |
| SYBETYPE   | EXINFO         | Invalid argument type given to Hyper/DB-Library.  |
| SYBEUCPT   | EXUSER         | Unrecognized custom-format parameter-type encountered in dbstrbuild.  |
| SYBEUCRR   | EXCONSISTENCY  | Internal software error: Unknown connection result reported by dbpasswd.  |
| SYBEUDTY   | EXCONSISTENCY  | Unknown datatype encountered.   |
| SYBEUFDS   | EXUSER         | Unrecognized format encountered in dbstrbuild.  |
| SYBEUFDT   | EXCONSISTENCY  | Unknown fixed-length datatype encountered.  |
| SYBEUHST   | EXUSER         | Unknown host machine name.  |
| SYBEUMSG   | EXCOMM         | Unknown message-id in MSG datastream.   |
| SYBEUNAM   | EXFATAL        | Unable to get current user name from operating system.  |
| SYBEUNOP   | EXNONFATAL     | Unknown option passed to dbsetopt.  |



| Error name | Error severity | Error text   |
|------------|----------------|--|
| SYBEUNT    | EXUSER         | Unknown network type found in interface file.  |
| SYBEURCI   | EXRESOURCE     | International Release: Unable to read copyright information from the DB-Library localization file.   |
| SYBEUREI   | EXRESOURCE     | International Release: Unable to read error information from the DB-Library localization file.   |
| SYBEUREM   | EXRESOURCE     | International Release: Unable to read error mnemonic from the DB-Library localization file.  |
| SYBEURES   | EXRESOURCE     | International Release: Unable to read error string from the DB-Library localization file.  |
| SYBEURMI   | EXRESOURCE     | International Release: Unable to read money-format information from the DB-Library localization file.  |
| SYBEUSCT   | EXCOMM         | Unable to set communications timer.  |
| SYBEUTDS   | EXCOMM         | Unrecognized TDS version received from the server.   |
| SYBEUVBF   | EXPROGRAM      | Attempt to read an unknown version of bcp format file.   |
| SYBEUVDT   | EXCONSISTENCY  | Unknown variable-length datatype encountered.  |
| SYBEVDPT   | EXUSER         | For bulk copy, all variable-length data must have either a length-prefix or a terminator specified.  |
| SYBEWAID   | EXCONSISTENCY  | DB-Library internal error: ALTFMT following ALTNAME has wrong id.  |
| SYBEWRIT   | EXCOMM         | Write to the server failed.  |
| SYBEXOCI   | EXNONFATAL     | International Release: A character-set translation overflowed its destination buffer while using bcp to copy data from a host-file to the server.  |
| SYBEXTDN   | EXPROGRAM      | Warning: The <i>xlt_todisp</i> parameter to <i>dbfree_xlate</i> was NULL. The space associated with the <i>xlt_tosrv</i> parameter has been freed. |

| Error name | Error severity | Error text   |
|------------|----------------|--|
| SYBEXTN    | EXPROGRAM      | The <i>xlt_tosrv</i> and <i>xlt_todisp</i> parameters to <i>dbfree_xlate</i> were NULL.  |
| SYBEXTSN   | EXPROGRAM      | Warning: The <i>xlt_tosrv</i> parameter to <i>dbfree_xlate</i> was NULL. The space associated with the <i>xlt_todisp</i> parameter has been freed. |
| SYBEZTXT   | EXINFO         | Attempt to send zero length text or image to dataserver using <i>dbwritetext</i> .   |
| UNUSED     | EXINFO         | This error number is unused.   |

Error severities

Table 2-33 lists the meanings for each symbolic error severity value.

**Table 2-33: Error severities**

| Error severity | Numerical equivalent | Explanation  |
|----------------|----------------------|--|
| EXINFO         | 1                    | Informational, non-error.  |
| EXUSER         | 2                    | User error.  |
| EXNONFATAL     | 3                    | Non-fatal error.   |
| EXCONVERSION   | 4                    | Error in DB-Library data conversion.   |
| EXSERVER       | 5                    | The Server has returned an error flag.   |
| EXTIME         | 6                    | We have exceeded our timeout period while waiting for a response from the Server—the DBPROCESS is still alive. |
| EXPROGRAM      | 7                    | Coding error in user program.  |
| EXRESOURCE     | 8                    | Running out of resources—the DBPROCESS may be dead.  |
| EXCOMM         | 9                    | Failure in communication with Server—the DBPROCESS is dead.  |
| EXFATAL        | 10                   | Fatal error—the DBPROCESS is dead.   |
| EXCONSISTENCY  | 11                   | Internal software error—notify Sybase Technical Support.   |

See also

DBDEAD, dberhandle

# Options

|             |  |
|-------------|--|
| Description | The complete list of DB-Library options.   |
| Syntax      | <pre>#include &lt;sybfront.h&gt;  #include &lt;sybdb.h&gt;</pre>   |
| Usage       | <ul style="list-style-type: none"> <li>dbsetopt and dbclopt use the following constants, defined in <i>sybdb.h</i>, for setting and clearing options. All options are off by default. These options are available:</li> <li>DBARITHABORT – If this option is set, the server will abort a query when an arithmetic exception occurs during its execution.</li> <li>DBARITHIGNORE – If this option is set, the server will substitute null values for selected or updated values when an arithmetic exception occurs during query execution. The Adaptive Server Enterprise will not return a warning message. If neither DBARITHABORT nor DBARITHIGNORE is set, Adaptive Server Enterprise will substitute null values and print a warning message after the query has been executed.</li> <li>DBAUTH – This option sets system administration authorization levels. Possible levels are: “sa”, “sso”, “oper”, and “dbcc_edit”. For information on these levels, see the <i>Adaptive Server Enterprise Reference Manual</i>.</li> <li>DBBUFFER – This option allows the application to buffer result rows, so that it can access them non-sequentially using the dbgetrow function. This option is handled locally by DB-Library and is not a server option. When the option is set, you supply a parameter that is the number of rows you want buffered. If you use 0 as the number of rows to buffer, the buffer will be set to a default size (currently 1000 rows).</li> </ul> <p>When an application calls dbclopt to clear the DBBUFFER option, DB-Library frees the memory associated with the row buffer.</p> <ul style="list-style-type: none"> <li>DBCHAINXACTS – This option is used to select chained or unchained transaction behavior.</li> </ul> <p>Chained behavior means that each SQL statement that modifies or retrieves data implicitly begins a multi-statement transaction. Any delete, insert, open, fetch, select, or update statement implicitly begins a transaction. An explicit commit or rollback statement is required to end the transaction. Chained mode provides compatibility with ANSI SQL.</p> |

Unchained behavior means that each SQL statement that modifies or retrieves data is implicitly a distinct transaction. Explicit begin transaction and commit or rollback statements are required to define a multi-statement transaction.

This option is off (indicating unchained behavior) by default. Applications that operate in chained mode should turn on the option right after a connection has been opened, since this option affects the behavior of all queries.

- **DBDATEFIRST** – Sets the first weekday to a number from 1 to 7. The `us_english` default is 1 (Sunday).
- **DBDATEFORMAT** – Sets the order of the date parts month/day/year for entering `datetime` or `smalldatetime` data. Valid arguments are “mdy,” “dmy,” “ymd,” “ydm,” “myd,” or “dym”. The `us_english` default is “mdy.”

Row buffering provides a way to keep a specified number of server result rows in program memory. Without row buffering, the result row generated by each new `dbnextrow` call overwrites the contents of the previous result row. Therefore, row buffering is useful for programs that need to look at result rows in a non-sequential manner. However, it does carry a memory and performance penalty because each row in the buffer must be allocated and freed individually. Therefore, use it only if you need to. Specifically, the application should only turn the **DBBUFFER** option on if it calls `dbgetrow` or `dbsetrow`. Note that row buffering has nothing to do with network buffering and is a completely independent issue. (See the `dbgetrow`, `dbnextrow`, and `dbclrbuf` reference pages.)

- **DBFIPSFLAG** – Setting this option causes the server to flag non-standard SQL commands. This option is off by default.
- **DBISOLATION** – This option is used to specify the transaction isolation level. Possible levels are 1 and 3. The default level is 1. Setting the level to 3 causes all pages of tables specified in a `select` query inside a transaction to be locked for the duration of the transaction.
- **DBNATLANG** – This is a DB-Library Internationalization option. Associate the specified **DBPROCESS** (or all open **DBPROCESS**es, if a **DBPROCESS** is not specified) with a national language. If the national language is not set for a particular **DBPROCESS**, U.S. English is used by default.

In programs that allow application users to make ad hoc queries, the user may override DBNATLANG with the Transact-SQL set language command.

---

**Note** All DBPROCESSes opened using a particular LOGINREC will also use that LOGINREC's associated national language. Use the DBSETLNATLANG macro to associate a national language with a LOGINREC.

---

- **DBNOAUTOFREE** – This option causes the command buffer to be cleared only by an explicit call to `dbfreebuf`. When **DBNOAUTOFREE** is not set, after a call to `dbsqlxexec` or `dbsqlsend` the first call to either `dbcmd` or `dbcmd` automatically clears the command buffer before the new text is entered.
- **DBNOCOUNT** – This option causes the server to stop sending back information about the number of rows affected by each SQL statement. The application can otherwise obtain this information by calling `DBCOUNT`.
- **DBNOEXEC** – If this option is set, the server will process the query through the compile step but the query will not be executed. This can be used in conjunction with `DBSHOWPLAN`.
- **DBOFFSET** – This option indicates that the server should return offsets for certain constructs in the query. **DBOFFSET** takes a parameter that specifies the particular construct. The valid parameters for this option are “select,” “from,” “table,” “order,” “compute,” “statement,” “procedure,” “execute,” or “param.” (Note that “param” refers to parameters of stored procedures.) Calls to routines such as `dbsetopt` can specify these option parameters in either lowercase or uppercase. Offsets are returned only if the batch contains no syntax errors.
- **DBPARSEONLY** – If this option is set, the server only checks the syntax of the query and returns error messages to the host. Offsets are returned if the **DBOFFSET** option is set and there are no errors.
- **DBPRCOLSEP** – Specify the column separator character(s). Query results rows formatted using `dbprhead`, `dbprrow`, `dbsprhead`, `dbsprline`, and `dbspr1row` will have columns separated by the specified string. The default separator is an ASCII 0x20 (space). The third parameter, a string, is not necessarily null-terminated. The length of the string used is given as the fourth parameter in the call to `dbsetopt`. To revert to using the default separator, specify a length of -1. In this case, the third parameter is ignored.

- **DBPRLINELEN** – Specify the maximum number of characters to be placed on one line. This value is used by `dbprhead`, `dbprrow`, `dbsprhead`, `dbsprline`, and `dbspr1row`. The default line length is 80 characters.
- **DBPRLINESEP** – Specify the row separator character to be used by `dbprhead`, `dbprrow`, `dbsprhead`, `dbsprline`, and `dbspr1row`. The default separator is a newline (ASCII 0x0D or 0x0A, depending on the host system). The third parameter, a string, is not necessarily null-terminated. The length of the string is given as the fourth parameter in the call to `dbsetopt`. To revert to the default terminator, specify a length of -1; in this case, the third parameter is ignored.
- **DBPRPAD** – Specify the pad character used when printing results using `dbprhead`, `dbprrow`, `dbsprhead`, `dbsprline`, and `dbspr1row`. To activate padding, specify `DBPADON` as the fourth parameter in the `dbsetopt` call. The pad character may be specified as the third parameter in the `dbsetopt` call. If the character is not specified, the ASCII character 0x20 (space) is used. To turn off padding, call `dbsetopt` with `DBPADOFF` as the fourth parameter; the third parameter is ignored when turning padding off.
- **DBROWCOUNT** – If this option is set to a value greater than 0, the server limits the number of regular rows returned for `select` statements and the number of table rows affected by `update` or `delete` statements. This option does not limit the number of compute rows returned by a `select` statement.  
  
**DBROWCOUNT** works somewhat differently from most options. It is always set on, never off. Setting **DBROWCOUNT** to 0 sets it back to the default – that is, to return all the rows generated by a `select` statement. Therefore, the way to turn **DBROWCOUNT** *off* is to set it *on* with a count of 0.
- **DBSHOWPLAN** – If this option is set, the server generates a description of the processing plan after compilation and continue executing the query.
- **DBSTAT** – This option determines when performance statistics (CPU time, elapsed time, I/O, and so on) will be returned to the host after each query. **DBSTAT** takes one of two parameters: “io”, for statistics about Adaptive Server Enterprise internal I/O; and “time”, for information about Adaptive Server Enterprise’s parsing, compilation, and execution times. These statistics are received by DB-Library in the form of informational messages, and application programs can access them through the user-supplied message handler.
- **DBSTORPROCID** – If this option is set, the server will send the stored procedure ID to the host before sending rows generated by the stored procedure.

- **DBTEXTLIMIT** – This option causes DB-Library to limit the size of returned text or image values. When setting this option, you supply a parameter that is the length, in bytes, of the longest text or image value that your program can handle. DB-Library will read but ignore any part of a text or image value that goes over this limit. DB-Library's default behavior is to read and return all the data sent by the server. To restore this default behavior, set DBTEXTLIMIT to a value less than 1. In the case of huge text values, it may take some time for the entire text value to be returned over the network. To keep the server from sending this extra text in the first place, use the DBTEXTSIZE option instead.
- **DBTEXTSIZE** – This option changes the value of the server global variable @@textsize, which limits the size of text or image values that the server returns. When setting this option, you supply a parameter that is the length, in bytes, of the longest text or image value that the server should return. @@textsize has a default value of 32,768 bytes. Note that, in programs that allow application users to make ad hoc queries, the user may override this option with the Transact-SQL set textsize command. To set a text limit that the user cannot override, use the DBTEXTLIMIT option instead.
- **DBBUFFER**, **DBNOAUTOFREE**, and **DBTEXTLIMIT** are DB-Library options. That is, they affect DB-Library but are not sent to the server. The other options are Adaptive Server Enterprise options – they are sent to the server. Adaptive Server Enterprise options can also be set through Transact-SQL commands.
- As mentioned in the preceding descriptions, certain options take parameters as shown in Table 2-34.

**Table 2-34: Parameter values for options**

| Option      | Possible parameter values   |
|-------------|---|
| DBTEXTSIZE  | “0” to “2,147,483,647”  |
| DBOFFSET    | “select”, “from”, “table”, “order”, “compute”, “statement”, “procedure”, “execute”, or “param”                |
| DBSTAT      | “io” or “time”  |
| DBROWCOUNT  | “0” to “2,147,483,647”  |
| DBBUFFER    | “0” to either “32,767” or “2,147,483,647”, depending on whether your <i>int</i> datatype is 2 or 4 bytes long |
| DBTEXTLIMIT | “0” to “2,147,483,647”  |

dbsetopt requires that an option parameter be specified when setting any option on the preceding list. dbclopt and dbisopt require that the parameter be specified only for DBOFFSET and DBSTAT. This is because DBOFFSET and DBSTAT are the only options that can have multiple settings at a time, and thus they require further definition before being cleared or checked.

Note that parameters specified in calls to dbsetopt, dbclopt, and dbisopt are always passed as character strings and must be quoted, even if they are numeric values. See the dbsetopt reference page.

See also dbclopt, dbisopt, dbsetopt

Types

|             |  |
|-------------|--|
| Description | Datatypes and symbolic constants for datatypes used by DB-Library.   |
| Syntax      | <pre>#include &lt;sybfront.h&gt;  #include &lt;sybdb.h&gt;</pre>   |
| Usage       | <ul style="list-style-type: none"><li>Table 2-35 lists the symbolic constants for server datatypes. dbconvert and dbwillconvert use these constants. In addition, the routines dbcoltype, dbalttype, and dbrettype will return one of these types.</li></ul> |



**Table 2-35: Symbolic constants for server datatypes**

| Symbolic constant | Represents  |
|-------------------|---|
| SYBDATETIME       | datetime type.  |
| SYBDATETIME4      | 4-byte datetime type.                                     |
| SYBMONEY4         | 4-byte money type.  |
| SYBMONEY          | money type.   |
| SYBFLT8           | 8-byte float type.  |
| SYBDECIMAL        | decimal type.   |
| SYBNUMERIC        | numeric type.   |
| SYBREAL           | 4-byte float type.  |
| SYBINT4           | 4-byte integer.   |
| SYBINT2           | 2-byte integer.   |
| SYBINT1           | 1-byte integer.   |
| SYBIMAGE          | image type.   |
| SYBTEXT           | text type.  |
| SYBCHAR           | char type.  |
| SYBBIT            | bit type.   |
| SYBBINARY         | binary type.  |
| SYBBOUNDARY       | Security sensitivity_boundary type.                       |
|                   | <b>Note</b> Use DBCHAR as the type for program variables. |
| SYBSENSITIVITY    | Security sensitivity type.                                |
|                   | <b>Note</b> Use DBCHAR as the type for program variables. |

See the *Adaptive Server Enterprise Transact-SQL Users Guide*.

- Here is a list of C datatypes used by DB-Library functions. These types are useful for defining program variables, particularly variables used with `dbbind`, `dbaltbind`, `dbconvert`, and `dbdatecrack`.

```
/* char, text, boundary, and sensitivity types */
typedef char          DBCHAR;

/* binary and image type */
typedef unsigned char DBBINARY;

/* 1-byte integer */
typedef unsigned char DBTINYINT;

/* 2-byte integer */
typedef short         DBSMALLINT;
```

```
/* unsigned 2-byte integer */
typedef unsigned short  DBUSMALLINT;

/* 4-byte integer */
typedef long            DBINT;

/* 4-byte float type */
typedef float          DBREAL;

typedef struct          dbnumeric
{
    char                precision;
    char                scale;
    unsigned char       val[MAXNUMLEN];
} DBNUMERIC;

typedef DBNUMERIC       DBDECIMAL;

/* 8-byte float type */
typedef double          DBFLT8;

/* bit type */
typedef unsigned char   DBBIT;

/* SUCCEED or FAIL */
typedef int             RETCODE;

/* datetime type */
typedef struct          datetime
{
    /* number of days since 1/1/1900 */
    long                dtdays;
    /* 300ths of a second since midnight */
    unsigned long       dttime;
} DBDATETIME;

/* 4-byte datetime type */
typedef struct          datetime4
{
    /* number of days since 1/1/1900 */
    unsigned short      numdays;
    /* number of minutes since midnight */
    unsigned short      nummins;
} DBDATETIME4;

typedef struct          dbdaterec
```

```

{
/* 1900 to the future */
long    dateyear;
/* 0 - 11 */
long    datemonth;
/* 1 - 31 */
long    datedmonth;
/* 1 - 366 */
long    datedyear;
/* 0 - 6 (day names depend on language */
long    datedweek;
/* 0 - 23 */
long    datehour;
/* 0 - 59 */
long    dateminute;
/* 0 - 59 */
long    datesecsecond;
/* 0 - 997 */
long    datemsecond;
/* 0 - 127 -- NOTE: Currently unused.*/
long    datetzone;
} DBDATEREC;

/* money type */
typedef struct          money
{
    long                mnyhigh;
    unsigned long       mnylow;
} DBMONEY;

/* 4-byte money type */
typedef signed long     DBMONEY4;

/* Pascal-type string */
typedef struct          dbvarychar
{
    /* character count */
    DBSMALLINT          len;
    /* non-terminated string */
    DBCHAR              str[DBMAXCHAR];
} DBVARYCHAR;

/* Pascal-type binary array */
typedef struct          dbvarybin
{
    /* byte count */

```

```
        DBSMALLINT      len;
        /* non-terminated array */
        BYTE            array[DBMAXCHAR];
    } DBVARYBIN;

    /* Used by DB-Library for indicator variables */
    typedef      DBSMALLINT      DBINDICATOR;
```

---

**Note** The SYBBOUNDARY and SYBSENSITIVITY symbolic constants correspond to the program variable type DBCHAR.

---

See also

dbaltbind, dbalttype, dbbind, dbcoltype, dbconvert, dbprtype, dbrettype, dbwillconvert, Options

# Bulk Copy Routines

This chapter describes the DB-Library bulk copy routines.

| Topic                      | Page |
|----------------------------|------|
| Introduction to bulk copy  | 417  |
| List of bulk copy routines | 420  |

## Introduction to bulk copy

Bulk copy is a tool for high-speed transfer of data between a database table and program variables or a host file. It provides an alternative to SQL insert and select commands.

The DB-Library/C bulk copy special library is a collection of routines that provide bulk copy functionality to a DB-Library/C application. A DB-Library/C application may find bulk copy useful if it needs to exchange data with a non-database application, load data into a new database, or move data from one database to another.

## Transferring data into the database

Data can be copied into a database from program variables or from a flat file on the client's host machine.

When you are copying data into a database table, the chief advantage of bulk copy over the alternative SQL insert command is speed. Also, SQL insert requires that the data be in character string format, while bulk copy can transfer native datatypes.

When copying data into a non-indexed table, the “high speed” version of bulk copy is used, which means that no data logging is performed during the transfer. If the system fails before the transfer is complete, no new data will remain in the database. Because high-speed transfer affects the recoverability of the database, it is only enabled if the Adaptive Server Enterprise option `select into/bulkcopy` has been turned on. If the option is not enabled, and a user tries to copy data into a table that has no indexes, Adaptive Server Enterprise generates an error message.

After the bulk copy is complete, the System Administrator should dump the database to ensure its future recoverability.

When you copy data into an indexed table, a slower version of `bcpl` is automatically used, and row inserts are logged.

To copy data into a database, a DB-Library/C application must perform the following introductory steps:

- Call `dblogin` to acquire a `LOGINREC` structure for later use with `dbopen`.
- Call `BCP_SETL` to set up the `LOGINREC` for bulk copy operations into the database.
- Call `dbopen` to establish a connection with Adaptive Server Enterprise.
- Call `bcpl_init` to initialize the bulk copy operation and inform Adaptive Server Enterprise whether the copy will be performed from program variables or from a host file. To copy data into the database, the `bcpl_init direction` parameter must be passed as `DB_IN`.

At this point, an application copying data from program variables will need to perform different steps than an application copying data from a host file.

To copy data from program variables, a DB-Library/C application must perform the following steps in addition to the introductory ones listed previously:

- Call `bcpl_bind` once for each program variable that is to be bound to a database column.
- Transfer a batch of data in a loop:
  - Assign program variables the data values to transfer.
  - Call `bcpl_sendrow` to send the row of data.
- After a batch of rows has been sent, call `bcpl_batch` to save the rows in Adaptive Server Enterprise.

- After all the data has been sent, call `bcp_done` to end the bulk copy operation.

To copy data from a host file, a DB-Library/C application needs to perform the following steps in addition to the introductory ones listed previously:

- Call `bcp_control` to set the batch size and change control parameter default settings.
- Call `bcp_columns` to set the total number of columns found in the host file.
- Call `bcp_colfmt` once for each column in the host file. If the host file matches the database table exactly, an application does not have to call `bcp_colfmt`.
- Call `bcp_exec` to start the copy in.

## Transferring data out of the database to a flat file

Data can be copied out from a database only into an operating system (host) file. Bulk copy does not allow the transfer of data from a database into program variables.

When transferring data out to a host file from a database table, the chief advantage of bulk copy over SQL select is that it allows very specific output file formats to be specified. Bulk copy is not significantly faster than SQL select.

To copy data out from a database, a DB-Library/C application must perform the following steps:

- 1 Call `dblogin` to acquire a `LOGINREC` structure for later use with `dbopen`.
- 2 Call `dbopen` to establish a connection with Adaptive Server Enterprise.
- 3 Call `bcp_init` to initialize the bulk copy operation. To copy data out from the database, *direction* must be passed as `DB_OUT`.
- 4 Call `bcp_control` to set the batch size and change control parameter default settings.
- 5 Call `bcp_columns` to set the total number of columns found in the host file.
- 6 Call `bcp_colfmt` once for each column in the host file. If the host file matches the database table exactly, an application does not have to call `bcp_colfmt`.
- 7 Call `bcp_exec` to start the copy out.

## List of bulk copy routines

| <b>Routine</b> | <b>Description</b>   |
|----------------|--|
| bcp_batch      | Save any preceding rows in Adaptive Server Enterprise.   |
| bcp_bind       | Bind data from a program variable to a Adaptive Server Enterprise table.   |
| bcp_colfmt     | Specify the format of a host file for bulk copy purposes.  |
| bcp_colfmt_ps  | Specify the format of a host file for bulk copy purposes, with precision and scale support for numeric and decimal columns.  |
| bcp_collen     | Set the program variable data length for the current bulk copy into the database.  |
| bcp_colptr     | Set the program variable data address for the current bulk copy into the database.   |
| bcp_columns    | Set the total number of columns found in the host file.  |
| bcp_control    | Change various control parameter default settings.   |
| bcp_done       | End a bulk copy from program variables into Adaptive Server Enterprise.  |
| bcp_exec       | Execute a bulk copy of data between a database table and a host file.  |
| bcp_getl       | Determine if the LOGINREC has been set for bulk copy operations.   |
| bcp_init       | Initialize bulk copy.  |
| bcp_moretext   | Send part of a text or image value to Adaptive Server Enterprise.  |
| bcp_options    | Set bulk copy options.   |
| bcp_readfmt    | Read a datafile format definition from a host file.  |
| bcp_sendrow    | Send a row of data from program variables to Adaptive Server Enterprise.   |
| BCP_SETL       | Set the LOGINREC for bulk copy operations into the database.   |
| bcp_setxlate   | Specify the character set translations to use when retrieving data from or inserting data into a Adaptive Server Enterprise. |
| bcp_writfmt    | Write a datafile format definition to a host file.   |



## bcp\_batch

|              |  |
|--------------|--|
| Description  | Save any preceding rows in Adaptive Server Enterprise.   |
| Syntax       | <pre>DBINT bcp_batch(dbproc)</pre> <pre>DBPROCESS *dbproc;</pre>   |
| Parameters   | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and Adaptive Server Enterprise.</p>   |
| Return value | The number of rows saved since the last call to bcp_batch, or -1 in case of error.   |
| Usage        | <ul style="list-style-type: none"> <li>When an application uses bcp_bind and bcp_sendrow to bulk-copy rows from program variables to Adaptive Server Enterprise tables, the rows are permanently saved in Adaptive Server Enterprise only when the program calls bcp_batch or bcp_done.</li> <li>You may call bcp_batch once every <i>n</i> rows or when there is a lull between periods of incoming data (as in a telemetry application). Of course, you may choose some other criteria, or may decide not to call bcp_batch at all. If bcp_batch is not called, the rows are permanently saved in Adaptive Server Enterprise when bcp_done is called.</li> <li>By default, Adaptive Server Enterprise copies all the rows specified in one batch. Adaptive Server Enterprise considers each batch to be a separate bcp operation. Each batch is copied in a single insert transaction, and if any row in the batch is rejected, the entire insert is rolled back. bcp then continues to the next batch. You can use bcp_batch to break large input files into smaller units of recoverability. For example, if 300,000 rows are bulk copied and bcp_batch is called every 100,000 rows, if there is a fatal error after row 200,000, the first two batches—200,000 rows—will have been successfully copied into Adaptive Server Enterprise.</li> <li>bcp_batch actually sends two commands to the server. The first command tells the server to permanently save the rows. The second tells the server to begin a new transaction. It is possible that the command to save the rows completes successfully but the command to start a new transaction does not. In this case, bcp_batch's error return of -1 does not indicate that the rows have not been successfully saved. To find out whether this has happened, an application can refer to the messages generated by Adaptive Server Enterprise or DB-Library/C.</li> </ul> |
| See also     | bcp_bind, bcp_done, bcp_sendrow  |

## bcp\_bind

|             |  |
|-------------|--|
| Description | Bind data from a program variable to an Adaptive Server Enterprise table.  |
| Syntax      | <pre>RETCODE bcp_bind (dbproc, varaddr, prefixlen, varlen,<br/>                  terminator, termilen, type,<br/>                  table_column)</pre><br><pre>DBPROCESS  *dbproc;<br/>BYTE       *varaddr;<br/>int         prefixlen;<br/>DBINT      varlen;<br/>BYTE       *terminator;<br/>int         termilen;<br/>int         type;<br/>int         table_column;</pre>  |
| Parameters  | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and Adaptive Server Enterprise.</p> <p><b>varaddr</b><br/>The address of the program variable from which the data will be copied. If type is SYBTEXT or SYBIMAGE, <i>varaddr</i> can be NULL. A NULL <i>varaddr</i> indicates that text and image values will be sent to Adaptive Server Enterprise in chunks by <i>bcp_moretext</i>, rather than all at once by <i>bcp_sendrow</i>.</p> <p><b>prefixlen</b><br/>The length, in bytes, of any length prefix this column may have. For example, strings in some non-C programming languages are made up of a one-byte length prefix, followed by the string data itself. If the data does not have a length prefix, set <i>prefixlen</i> to 0.</p> |

**varlen**

The length of the data in the program variable, *not* including the length of any length prefix and/or terminator. Setting *varlen* to 0 signifies that the data is null. Setting *varlen* to -1 indicates that the system should ignore this parameter.

For fixed-length datatypes, such as integer, the datatype itself indicates to the system the length of the data. Therefore, for fixed-length datatypes, *varlen* must always be -1, except when the data is null, in which case *varlen* must be 0.

For char, text, binary, and image datatypes, *varlen* can be -1, 0, or some positive value. If *varlen* is -1, the system will use either a length prefix or a terminator sequence to determine the length. (If both are supplied, the system will use the one that results in the shortest amount of data being copied.) If *varlen* is -1 and neither a prefix length nor a terminator sequence is specified, the system will return an error message. If *varlen* is 0, the system assumes the data is null. If *varlen* is some positive value, the system uses *varlen* as the data length. However, if, in addition to a positive *varlen*, a prefix length and/or terminator sequence is provided, the system determines the data length by using the method that results in the shortest amount of data being copied.

**terminator**

A pointer to the byte pattern, if any, that marks the end of this program variable. For example, C strings usually have a 1-byte terminator whose value is 0. If there is no terminator for the variable, set *terminator* to NULL.

If you want to designate the C null terminator as the program variable terminator, the simplest way is to use an empty string ("") as *terminator* and set *termLen* to 1, since the null terminator constitutes a single byte. For instance, the second `bcp_bind` call in the “Example” section below uses two tabs as the program variable terminator. It could be rewritten to use a C null terminator instead, as follows:

```
bcp_bind (dbproc, co_name, 0, -1, "", 1, 0, 2)
```

**termLen**

The length of this program variable’s terminator, if any. If there is no terminator for the variable, set *termLen* to 0.

### type

The datatype of your program variable, expressed as an Adaptive Server Enterprise datatype. The data in the program variable will be automatically converted to the type of the database column. If this parameter is 0, no conversion will be performed. See the dbconvert reference page for a list of supported conversions. That reference page also contains a list of Adaptive Server Enterprise datatypes.

### table\_column

The column in the database table to which the data will be copied. Column numbers start at 1.

**Return value** SUCCEED or FAIL.

**Examples**

- The following program fragment illustrates bcp\_bind:

```

LOGINREC          *login;
DBPROCESS         *dbproc;
char              co_name[MAXNAME];
DBINT             co_id;
DBINT             rows_sent;
DBBOOL           more_data;
char              *terminator = "\t\t";

/* Initialize DB-Library. */
if (dbinit() == FAIL)
    exit(ERREXIT);

/* Install error-handler and message-handler. */
dberrhandle(err_handler);
dbmsgghandle(msg_handler);

/* Open a DBPROCESS. */
login = dblogin();
BCP_SETL(login, TRUE);
dbproc = dbopen(login, NULL);

/* Initialize bcp. */
if (bcp_init(dbproc, "comdb..accounts_info",
            NULL, NULL, DB_IN) == FAIL)
    exit(ERREXIT);

/* Bind program variables to table columns. */
if (bcp_bind(dbproc, &co_id, 0, -1,
            (BYTE *)NULL, 0, 0, 1) == FAIL)
{
    fprintf(stderr, "bcp_bind, column 1, failed.\n");
}

```

```

        exit(ERREXIT);
    }

    if (bcp_bind
        (dbproc, co_name, 0, -1, (BYTE *)terminator,
         strlen(terminator), 0, 2)
        == FAIL)
    {
        fprintf(stderr, "bcp_bind, column 2, failed.\n");
        exit(ERREXIT);
    }

    while (TRUE)
    {
        /* Process/retrieve program data. */
        more_data = getdata(&co_id, co_name);

        if (more_data == FALSE)
            break;

        /* Send the data. */
        if (bcp_sendrow(dbproc) == FAIL)
            exit(ERREXIT);
    }

    /* Terminate the bulk copy operation. */
    if ((rows_sent = bcp_done(dbproc)) == -1)
        printf("Bulk-copy unsuccessful.\n");
    else
        printf("%ld rows copied.\n", rows_sent);

```

#### Usage

- There may be times when you want to copy data directly from a program variable into a table in Adaptive Server Enterprise, without having to first place the data in a host file or use the SQL insert command. The `bcp_bind` function is a fast and efficient way to do this.
- You must call `bcp_init` before calling this or any other bulk copy functions.
- There must be a separate `bcp_bind` call for every column in the Adaptive Server Enterprise table into which you want to copy. After the necessary `bcp_bind` calls have been made, you then call `bcp_sendrow` to send a row of data from your program variables to Adaptive Server Enterprise. The table to be copied into is set by calling `bcp_init`.
- You can override the program variable data length (*varlen*) for a particular column on the current copy in by calling `bcp_colln`.

- Whenever you want Adaptive Server Enterprise to checkpoint the rows already received, call `bcp_batch`. For example, you may want to call `bcp_batch` once for every 1000 rows inserted, or at any other interval.
- When there are no more rows to be inserted, call `bcp_done`. Failure to do so will result in an error.
- When using `bcp_bind`, the host file name parameter used in the call to `bcp_init`, *hfile*, must be set to NULL, and the direction parameter, *direction*, must be set to DB\_IN.
- Prefix lengths should not be used with fixed-length datatypes, such as integer or float. For fixed-length datatypes, since bulk copy can figure out the length of the data from the datatype, pass *prefixlen* as 0 and *varlen* as -1, except when the data is NULL, in which case *varlen* must be 0.
- Control parameter settings, specified with `bcp_control`, have no effect on `bcp_bind` row transfers.
- It is an error to call `bcp_columns` when using `bcp_bind`.

See also

`bcp_batch`, `bcp_colfmt`, `bcp_colln`, `bcp_colptr`, `bcp_columns`, `bcp_control`, `bcp_done`, `bcp_exec`, `bcp_init`, `bcp_moretext`, `bcp_sendrow`

## bcp\_colfmt

Description

Specify the format of a host file for bulk copy purposes.

Syntax

```
RETCODE bcp_colfmt (dbproc, host_colnum, host_type,  
                    host_prefixlen, host_colln,  
                    host_term, host_termlen,  
                    table_colnum)
```

```
DBPROCESS  *dbproc;  
int         host_colnum;  
int         host_type;  
int         host_prefixlen;  
DBINT      host_colln;  
BYTE       *host_term;  
int         host_termlen;  
int         table_colnum;
```

**Parameters****dbproc**

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and Adaptive Server Enterprise.

**host\_colnum**

The column in the host file whose format is being specified. The first column is number 1.

**host\_type**

The datatype of this column in the host file, expressed as an Adaptive Server Enterprise datatype. If it is different from the datatype of the corresponding column in the database table (*table\_colnum*), the conversion will be performed automatically. See the dbconvert reference page for a table of allowable data conversions. That reference page also contains a list of Adaptive Server Enterprise datatypes.

If you want to specify the same datatype as in the corresponding column of the database table (*table\_colnum*), this parameter should be set to 0.

---

**Note** bcp\_colfmt does not offer precision and scale support for numeric and decimal types. When setting the format of a numeric or decimal host column, bcp\_colfmt uses a default precision and scale of 18 and 0, respectively. To specify a different precision and scale, an application can call bcp\_colfmt\_ps.

---

**host\_prefixlen**

The length of the length prefix for this column in the host file. Legal prefix lengths are 1, 2, and 4 bytes. To avoid using a length prefix, this parameter should be set to 0. To let bcp decide whether to use a length prefix, this parameter should be set to -1. In such a case, bcp will use a length prefix (of whatever length is necessary) if the database column length is variable.

If more than one means of specifying a host file column length is used (such as a length prefix and a maximum column length, or a length prefix and a terminator sequence), bcp will look at all of them and use the one that results in the smallest amount of data being copied.

One valuable use for length prefixes is to simplify the specifying of null data values in a host file. For instance, assume you have a 1-byte length prefix for a 4-byte integer column. Ordinarily, the length prefix will contain a value of 4, to indicate that a 4-byte value follows. However, if the value of the column is NULL, the length prefix can be set to 0 to indicate that 0 bytes follow for the column.

*host\_collen*

The maximum length of this column's data in the host file, *not* including the length of any length prefix and/or terminator. Setting *host\_collen* to 0 signifies that the data is NULL. Setting *host\_collen* to -1 indicates that the system should ignore this parameter (that is, there is no default maximum length).

For fixed-length datatypes, such as integer, the length of the data is constant, except for the special case of null values. Therefore, for fixed-length datatypes, *host\_collen* must always be -1, except when the data is null, in which case *host\_collen* must be 0.

For char, text, binary, and image datatypes, *host\_collen* can be -1, 0, or some positive value. If *host\_collen* is -1, the system will use either a length prefix or a terminator sequence to determine the length of the data. (If both are supplied, the system will use the one that results in the shortest amount of data being copied.) If *host\_collen* is -1 and neither a prefix length nor a terminator sequence is specified, the system will return an error message. If *host\_collen* is 0, the system assumes the data is NULL. If *host\_collen* is some positive value, the system uses *host\_collen* as the maximum data length. However, if, in addition to a positive *host\_collen*, a prefix length and/or terminator sequence is provided, the system determines the data length by using the method that results in the shortest amount of data being copied.

*host\_term*

The terminator sequence to be used for this column. This parameter is mainly useful for char, text, binary, and image datatypes, because all other datatypes are of fixed length. To avoid using a terminator, set this parameter to NULL. To set the terminator to the NULL character, set *host\_term* to “\0”. To make the tab character the terminator, set *host\_term* to “\t”. To make the newline character the terminator, set *host\_term* to “\n”.

If more than one means of specifying a host file column length is used (such as a terminator and a length prefix, or a terminator and a maximum column length), bcp will look at all of them and use the one that results in the smallest amount of data being copied.

*host\_termlen*

The length, in bytes, of the terminator sequence to be used for this column. To avoid using a terminator, set this value to -1.

*table\_colnum*

The corresponding column in the database table. If this value is 0, this column will *not* be copied. The first column is column 1.



|              |   |
|--------------|---|
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"> <li>• <code>bcp_colfmt</code> allows you to specify the host file format for bulk copies. For bulk copy purposes, a format contains the following parts: <ul style="list-style-type: none"> <li>• A mapping from host file columns to database columns</li> <li>• The datatype of each host file column</li> <li>• The length of the optional length prefix of each column</li> <li>• The maximum length of the host file column's data</li> <li>• The optional terminating byte sequence for each column</li> <li>• The length of this optional terminating byte sequence</li> </ul> </li> <li>• Each call to <code>bcp_colfmt</code> specifies the format for one host file column. For example, if you have a table with five columns and want to change the default settings for three of those columns, you should first call <code>bcp_columns(dbproc, 5)</code>, and then call <code>bcp_colfmt</code> five times, with three of those calls setting your custom format. The remaining two calls should have their <i>host_type</i> set to 0, and their <i>host_prefixlen</i>, <i>host_colln</i>, and <i>host_termlen</i> parameters set to -1. The result of this would be to copy all five columns—three with your customized format and two with the default format.</li> <li>• <code>bcp_columns</code> <i>must</i> be called before any calls to <code>bcp_colfmt</code>.</li> <li>• You must call <code>bcp_colfmt</code> for every column in the host file, regardless of whether some of those columns use the default format or are skipped.</li> <li>• To skip a column, set the <i>table_column</i> parameter to 0.</li> </ul> |
| See also     | <code>bcp_batch</code> , <code>bcp_bind</code> , <code>bcp_colfmt_ps</code> , <code>bcp_colln</code> , <code>bcp_colptr</code> , <code>bcp_columns</code> , <code>bcp_control</code> , <code>bcp_done</code> , <code>bcp_exec</code> , <code>bcp_init</code> , <code>bcp_sendrow</code>   |

## bcp\_colfmt\_ps

|             |  |
|-------------|--|
| Description | Specify the format of a host file for bulk copy purposes, with precision and scale support for numeric and decimal columns.  |
| Syntax      | <pre>RETCODE bcp_colfmt_ps (dbproc, host_colnum, host_type,                         host_prefixlen, host_colln,                         host_term, host_termlen,                         table_colnum, typeinfo)</pre> |

```
DBPROCESS  *dbproc;  
int         host_colnum;  
int         host_type;  
int         host_prefixlen;  
DBINT       host_collen;  
BYTE        *host_term;  
int         host_termlen;  
int         table_colnum;  
DBTYPEINFO  *typeinfo;
```

---

**Note** bcp\_colfmt\_ps's parameters are identical to bcp\_colfmt's, except that bcp\_colfmt\_ps has the additional parameter *typeinfo*, which contains information about precision and scale for numeric or decimal columns.

---

## Parameters

### dbproc

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and Adaptive Server Enterprise.

### host\_colnum

The column in the host file whose format is being specified. The first column is number 1.

### host\_type

The datatype of this column in the host file, expressed as an Adaptive Server Enterprise datatype. If it is different from the datatype of the corresponding column in the database table (*table\_colnum*), the conversion will be performed automatically. See the dbconvert reference page for a table of allowable data conversions. That reference page also contains a list of Adaptive Server Enterprise datatypes.

If you want to specify the same datatype as in the corresponding column of the database table (*table\_colnum*), this parameter should be set to 0.

**host\_prefixlen**

The length of the length prefix for this column in the host file. Legal prefix lengths are 1, 2, and 4 bytes. To avoid using a length prefix, this parameter should be set to 0. To let bcp decide whether to use a length prefix, this parameter should be set to -1. In such a case, bcp will use a length prefix (of whatever length is necessary) if the database column length is variable.

If more than one means of specifying a host file column length is used (such as a length prefix and a maximum column length, or a length prefix and a terminator sequence), bcp will look at all of them and use the one that results in the shortest amount of data being copied.

One valuable use for length prefixes is to simplify the specifying of null data values in a host file. For instance, assume you have a 1-byte length prefix for a 4-byte integer column. Ordinarily, the length prefix will contain a value of 4, to indicate that a 4-byte value follows. However, if the value of the column is null, the length prefix can be set to 0, to indicate that 0 bytes follow for the column.

**host\_collen**

The maximum length of this column's data in the host file, not including the length of any length prefix and/or terminator. Setting *host\_collen* to 0 signifies that the data is NULL. Setting *host\_collen* to -1 indicates that the system should ignore this parameter (that is, there is no default maximum length).

For fixed-length datatypes, such as integer, the length of the data is constant, except for the special case of null values. Therefore, for fixed-length datatypes, *host\_collen* must always be -1, except when the data is NULL, in which case *host\_collen* must be 0.

For char, text, binary, and image datatypes, *host\_collen* can be -1, 0, or some positive value. If *host\_collen* is -1, the system will use either a length prefix or a terminator sequence to determine the length of the data. (If both are supplied, the system will use the one that results in the smallest amount of data being copied.) If *host\_collen* is -1 and neither a prefix length nor a terminator sequence is specified, the system will return an error message. If *host\_collen* is 0, the system assumes the data is NULL. If *host\_collen* is some positive value, the system uses *host\_collen* as the maximum data length. However, if, in addition to a positive *host\_collen*, a prefix length and/or terminator sequence is provided, the system determines the data length by using the method that results in the smallest amount of data being copied.

**host\_term**

The terminator sequence to be used for this column. This parameter is mainly useful for char, text, binary, and image datatypes, because all other types are of fixed length. To avoid using a terminator, set this parameter to NULL. To set the terminator to the null character, set *host\_term* to “\0”. To make the tab character the terminator, set *host\_term* to “\t”. To make the newline character the terminator, set *host\_term* to “\n”.

If more than one means of specifying a host file column length is used (such as a terminator and a length prefix, or a terminator and a maximum column length), bcp will look at all of them and use the one that results in the smallest amount of data being copied.

**host\_termlen**

The length, in bytes, of the terminator sequence to be used for this column. To avoid using a terminator, set this value to -1.

**table\_colnum**

The corresponding column in the database table. If this value is 0, this column will not be copied. The first column is column 1.

**typeinfo**

A pointer to a DBTYPEINFO structure containing information about the precision and scale of decimal or numeric host file columns. An application sets a DBTYPEINFO structure with values for precision and scale before calling bcp\_colfmt\_ps to specify the host file format of decimal or numeric columns.

If *typeinfo* is NULL, bcp\_colfmt\_ps is the equivalent of bcp\_colfmt. That is:

- If the server column is of type numeric or decimal, bcp\_colfmt\_ps picks up precision and scale values from the column.
- If the server column is not numeric or decimal, bcp\_colfmt\_ps uses a default precision of 18 and a default scale of 0.

If *host\_type* is not 0, SYBDECIMAL or SYBNUMERIC, *typeinfo* is ignored.

If *host\_type* is 0 and the corresponding server column is not numeric or decimal, *typeinfo* is ignored.

A DBTYPEINFO structure is defined as follows:

```
typedef struct typeinfo {  
    DBINT    precision;  
    DBINT    scale;
```

```
} DBTYPEINFO;
```

Legal values for *precision* are from 1 to 77. Legal values for *scale* are from 0 to 77. *scale* must be less than or equal to *precision*.

Return value

SUCCEED or FAIL.

Usage

- `bcp_colfmt_ps` is the equivalent of `bcp_colfmt`, except that `bcp_colfmt_ps` provides precision and scale support for numeric and decimal datatypes, which `bcp_colfmt` does not. Calling `bcp_colfmt` is equivalent to calling `bcp_colfmt_ps` with *typeinfo* as NULL.
- `bcp_colfmt_ps` allows you to specify the host file format for bulk copies. For bulk copy purposes, a format contains the following parts:
  - A mapping from host file columns to database columns
  - The datatype of each host file column
  - The length of the optional length prefix of each column
  - The maximum length of the host file column's data
  - The optional terminating byte sequence for each column
  - The length of this optional terminating byte sequence
- Each call to `bcp_colfmt_ps` specifies the format for one host file column. For example, if you have a table with five columns, and want to change the default settings for three of those columns, you should first call `bcp_columns(dbproc, 5)`, and then call `bcp_colfmt_ps` five times, with three of those calls setting your custom format. The remaining two calls should have their *host\_type* set to 0, and their *host\_prefixlen*, *host\_colln*, and *host\_termnlen* parameters set to -1. The result of this would be to copy all five columns—three with your customized format and two with the default format.
- `bcp_columns` *must* be called before any calls to `bcp_colfmt_ps`.
- You must call `bcp_colfmt_ps` for every column in the host file, regardless of whether some of those columns use the default format or are skipped.
- To skip a column, set the *table\_column* parameter to 0.

See also

`bcp_batch`, `bcp_bind`, `bcp_colfmt`, `bcp_colln`, `bcp_colptr`, `bcp_columns`, `bcp_control`, `bcp_done`, `bcp_exec`, `bcp_init`, `bcp_sendrow`

## bcp\_collen

**Description** Set the program variable data length for the current bulk copy into the database.

**Syntax** RETCODE bcp\_collen(dbproc, varlen, table\_column)

```
DBPROCESS    *dbproc;  
DBINT        varlen;  
int          table_column;
```

**Parameters**

**dbproc**

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and Adaptive Server Enterprise.

**varlen**

The length of the program variable, which does *not* include the length of the length prefix or terminator. Setting *varlen* to 0 signifies that the data is NULL. Setting it to -1 signifies that the data is variable-length and that the length will be determined by the length prefix or terminator. If both a length prefix and a terminator exist, bcp will use the one that results in the smallest amount of data being copied.

**table\_column**

The column in the Adaptive Server Enterprise table to which the data will be copied. Column numbers start at 1.

**Return value** SUCCEED or FAIL.

**Usage**

- The bcp\_collen function allows you to change the program variable data length for a particular column while running a copy in through calls to bcp\_bind.
- Initially, the program variable data length is determined when bcp\_bind is called. If the program variable data length changes between calls to bcp\_sendrow, and no length prefix or terminator is being used, you may call bcp\_collen to reset the length. The next call to bcp\_sendrow will use the length you just set.
- There must be a separate bcp\_collen call for every column in the table whose data length you want to modify.

**See also** bcp\_bind, bcp\_colptr, bcp\_sendrow

## bcp\_colptr

|              |   |
|--------------|---|
| Description  | Set the program variable data address for the current bulk copy into the database.  |
| Syntax       | <pre>RETCODE bcp_colptr(dbproc, colptr, table_column)</pre> <pre>DBPROCESS    *dbproc; BYTE          *colptr; int           table_column;</pre>   |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and Adaptive Server Enterprise.</p> <p><b>colptr</b><br/>The address of the program variable.</p> <p><b>table_column</b><br/>The column in the Adaptive Server Enterprise table to which the data will be copied. Column numbers start at 1.</p>   |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"> <li>• The <code>bcp_colptr</code> function allows you to change the program variable data address for a particular column while running a copy in through calls to <code>bcp_bind</code>.</li> <li>• Initially, the program variable data address is determined when <code>bcp_bind</code> is called. If the program variable data address changes between calls to <code>bcp_sendrow</code>, you may call <code>bcp_colptr</code> to reset the address of the data. The next call to <code>bcp_sendrow</code> will use the data at the address you just set.</li> <li>• There must be a separate <code>bcp_colptr</code> call for every column in the table whose data address you want to modify.</li> </ul> |
| See also     | <code>bcp_bind</code> , <code>bcp_colln</code> , <code>bcp_sendrow</code>   |

## bcp\_columns

|             |   |
|-------------|---|
| Description | Set the total number of columns found in the host file. |
| Syntax      | <pre>RETCODE bcp_columns(dbproc, host_colcount)</pre>   |

|              |  |
|--------------|--|
|              | DBPROCESS    *dbproc;<br>int            host_colcount;   |
| Parameters   | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and Adaptive Server Enterprise.</p> <p>host_colcount</p> <p>The total number of columns in the host file. Even if you are preparing to bulk copy data from the host file to an Adaptive Server Enterprise table and do not intend to copy all columns in the host file, you must still set <i>host_colcount</i> to the total number of host file columns.</p> |
| Return value | SUCCEED or FAIL.   |
| Usage        | <ul style="list-style-type: none"><li>• This function sets the total number of columns found in a host file for use with bulk copy. This routine may be called only after bcp_init has been called with a valid file name.</li><li>• You should call this routine only if you intend to use a host file format that differs from the default. The default host file format is described on the bcp_init reference page.</li><li>• After calling bcp_columns, you must call bcp_colfmt <i>host_colcount</i> times, because you are defining a completely custom file format.</li></ul>  |
| See also     | bcp_colfmt, bcp_init   |

## bcp\_control

|             |  |
|-------------|--|
| Description | Change various control parameter default settings.   |
| Syntax      | RETCODE bcp_control(dbproc, field, value)  |
|             | DBPROCESS    *dbproc;<br>int            field;<br>DBINT         value;   |
| Parameters  | <p>dbproc</p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and Adaptive Server Enterprise.</p> |



field

A control-parameter identifier consisting of one of the following symbolic values:

| Field      | Description   |
|------------|---|
| BCPMAXERRS | The number of errors allowed before giving up. The default is 10.   |
| BCPFIRST   | The first row to copy. The default is 1. A value of less than 1 resets this field to its default value of 1.  |
| BCPLAST    | The last row to copy. The default is to copy all rows. A value of less than 1 resets this field to its default value.   |
| BCPBATCH   | The number of rows per batch. The default is 0, which means that the entire bulk copy will be done in one batch. This field is only meaningful when copying from a host file into Adaptive Server Enterprise. |

value

The value to change the corresponding control parameter to.

Return value

SUCCEED or FAIL.

Usage

- This function sets various control parameters for bulk copy operations, including the number of errors allowed before aborting a bulk copy, the numbers of the first and last rows to copy, and the batch size.
- These control parameters are only meaningful when the application copies between a host file and an Adaptive Server Enterprise table. Control parameter settings have no effect on bcp\_bind row transfers.
- By default, Adaptive Server Enterprise copies all the rows specified in one batch. Adaptive Server Enterprise considers each batch to be a separate bcp operation. Each batch is copied in a single insert transaction, and if any row in the batch is rejected, the entire insert is rolled back. bcp then continues to the next batch. You can use bcp\_batch to break large input files into smaller units of recoverability. For example, if 300,000 rows are bulk copied in with a batch size of 100,000 rows, and there is a fatal error after row 200,000, the first two batches—200,000 rows—will have been successfully copied into Adaptive Server Enterprise. If batching had not been used, no rows would have been copied into Adaptive Server Enterprise.
- The following program fragment illustrates bcp\_control:

```
LOGINREC      *login;
DBPROCESS     *dbproc;
DBINT         rowsread;
```

```
/* Initialize DB-Library. */
if (dbinit() == FAIL)
    exit(ERREXIT);

/* Install error-handler and message-handler. */
dberrhandle(err_handler);
dbmsghandle(msg_handler);

/* Open a DBPROCESS. */
login = dblogin();
BCP_SETL(login, TRUE);
dbproc = dbopen(login, NULL);

/* Initialize bcp. */
if (bcp_init(dbproc, "comdb..address", "address.add",
    "addr.error", DB_IN) == FAIL)
    exit(ERREXIT);

/* Set the number of rows per batch. */
if (bcp_control(dbproc, BCPBATCH, 1000) == FAIL)
{
    printf("bcp_control failed to set batching behavior.\n");
    exit(ERREXIT);
}

/* Set host column count. */
if (bcp_columns(dbproc, 1) == FAIL)
{
    printf("bcp_columns failed.\n");
    exit(ERREXIT);
}

/* Set the host-file format. */
if (bcp_colfmt(dbproc, 1, 0, 0, -1, (BYTE *)("\n"), 1, 1) == FAIL)
{
    printf("bcp_colformat failed.\n");
    exit(ERREXIT);
}

/* Now, execute the bulk copy. */
if (bcp_exec(dbproc, &rowsread) == FAIL)
{
    printf("Incomplete bulk copy. Only %ld row%c copied.\n",
        rowsread, (rowsread == 1) ? ' ' : 's');

    exit(ERREXIT);
}
```

```
}
```

See also `bcp_batch`, `bcp_bind`, `bcp_colfmt`, `bcp_colln`, `bcp_colptr`, `bcp_columns`,  
`bcp_done`, `bcp_exec`, `bcp_init`

## **bcp\_done**

**Description** End a bulk copy from program variables into Adaptive Server Enterprise.

**Syntax** `DBINT bcp_done(dbproc)`

`DBPROCESS*dbproc;`

**Parameters** `dbproc`

A pointer to the `DBPROCESS` structure that provides the connection for a particular front-end/Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and Adaptive Server Enterprise.

**Return value** The number of rows permanently saved since the last call to `bcp_batch`, or -1 in case of error.

**Usage** `bcp_done` ends a bulk copy performed with `bcp_bind` and `bcp_sendrow`. It should be called after the last call to `bcp_sendrow` or `bcp_moretext`. Failure to call `bcp_done` after you have completed copying in all your data will result in unpredictable errors.

See also `bcp_batch`, `bcp_bind`, `bcp_moretext`, `bcp_sendrow`

## **bcp\_exec**

**Description** Execute a bulk copy of data between a database table and a host file.

**Syntax** `RETCODE bcp_exec(dbproc, rows_copied)`

`DBPROCESS *dbproc;`  
`DBINT *rows_copied;`

|              |  |
|--------------|--|
| Parameters   | dbproc   |
|              | A pointer to the DBPROCESS structure that provides the connection for a particular front-end/Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and Adaptive Server Enterprise.  |
|              | rows_copied  |
|              | A pointer to a DBINT. bcp_exec will fill this DBINT with the number of rows successfully copied. If set to NULL, this parameter will not be filled in by bcp_exec.   |
| Return value | SUCCEED or FAIL.   |
|              | bcp_exec returns SUCCEED if all rows are copied. If a partial or complete failure occurs, bcp_exec returns FAIL. Check the <i>rows_copied</i> parameter for the number of rows successfully copied.  |
| Usage        | <ul style="list-style-type: none"><li>• This routine copies data from a host file to a database table or vice-versa, depending on the value of the <i>direction</i> parameter in bcp_init.</li><li>• Before calling this function you must call bcp_init with a valid host file name. Failure to do so will result in an error.</li><li>• The following program fragment illustrates bcp_exec:</li></ul> |

```
LOGINREC      *login;
DBPROCESS     *dbproc;
DBINT         rowsread;

/* Initialize DB-Library. */
if (dbinit() == FAIL)
    exit(ERREXIT);

/* Install error-handler and message-handler. */
dberrhandle(err_handler);
dbmsghandle(msg_handler);

/* Open a DBPROCESS. */
login = dblogin();
BCP_SETL(login, TRUE);
dbproc = dbopen(login, NULL);

/* Initialize bcp. */
if (bcp_init(dbproc, "pubs2..authors", "authors.save",
    (BYTE *)NULL, DB_OUT) == FAIL)
    exit(ERREXIT);

/* Now, execute the bulk copy. */
```

```
if (bcp_exec(dbproc, &rowsread) == FAIL)
    printf("Incomplete bulk copy. Only %ld row%c copied.\n",
        rowsread, (rowsread == 1) ? ' ' : 's');
```

See also `bcp_batch`, `bcp_bind`, `bcp_colfmt`, `bcp_collen`, `bcp_colptr`, `bcp_columns`, `bcp_control`, `bcp_done`, `bcp_init`, `bcp_sendrow`

## bcp\_getl

|              |  |
|--------------|--|
| Description  | Determine if the LOGINREC has been set for bulk copy operations.   |
| Syntax       | DBBOOL bcp_getl(loginrec)  |
|              | LOGINREC      *loginrec;   |
| Parameters   | loginrec<br>A pointer to a LOGINREC structure that will be passed as an argument to dbopen. You can get a LOGINREC structure by calling dblogin.   |
| Return value | “TRUE” or “FALSE.”   |
| Usage        | <ul style="list-style-type: none"><li>• <code>bcp_getl</code> returns “TRUE” if *<i>loginrec</i> is enabled for bulk copy operations, and “FALSE” if it is not.</li><li>• A DBPROCESS connection cannot be used for bulk copy in operations unless the LOGINREC used to open the connection has been set to allow bulk copy. The macro BCP_SETL sets a LOGINREC to allow bulk copy. By default, DBPROCESSes are not enabled for bulk copy operations.</li><li>• Applications that allow users to make ad hoc queries may want to avoid calling BCP_SETL (or call it with a value of “false” for the <i>enable</i> parameter) to prevent users from initiating a bulk copy sequence through SQL commands. Once a bulk copy sequence has begun, it cannot be stopped by an ordinary SQL command.</li><li>• If LOGINREC is NULL, <code>bcp_getl</code> returns “FALSE.”</li></ul> |
| See also     | <code>bcp_init</code> , <code>BCP_SETL</code> , <code>dblogin</code> , <code>dbopen</code>   |

## bcp\_init

|             |                       |
|-------------|-----------------------|
| Description | Initialize bulk copy. |
|-------------|-----------------------|

|              |   |
|--------------|---|
| Syntax       | <pre>RETCODE bcp_init(dbproc, tblname, hfile, errfile,                   direction)</pre> <pre>DBPROCESS  *dbproc; char        *tblname; char        *hfile; char        *errfile; int         direction;</pre>   |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and Adaptive Server Enterprise.</p> <p><b>tblname</b><br/>The name of the database table to be copied in or out. This name may also include the database name or the owner name. For example, pubs2.gracie.titles, pubs2.titles, gracie.titles, and titles are all legal table names.</p> <p><b>hfile</b><br/>The name of the host file to be copied in or out. If no host file is involved (the situation when data is being copied directly from variables), <i>hfile</i> should be NULL.</p> <p><b>errfile</b><br/>The name of the error file to be used. This error file will be filled with progress messages, error messages, and copies of any rows that, for any reason, could not be copied from a host file to an Adaptive Server Enterprise table.<br/><br/>If <i>errfile</i> is NULL, no error file is used.<br/><br/>If <i>hfile</i> is NULL, <i>errfile</i> is ignored. This is because an error file is not necessary when bulk-copying from program variables.</p> <p><b>direction</b><br/>The direction of the copy. It must be one of two values—DB_IN or DB_OUT. DB_IN indicates a copy from the host into the database table, while DB_OUT indicates a copy from the database table into the host file.<br/><br/>It is illegal to request a bulk copy from the database table (DB_OUT) without supplying a host file name.</p> |
| Return value | SUCCEED or FAIL.  |

## Usage

- `bcp_init` performs the necessary initialization for a bulk copy of data between the front-end and an Adaptive Server Enterprise. It sets the default host file data formats and examines the structure of the database table.
- `bcp_init` must be called before any other bulk copy functions. Failure to do so will result in an error.
- If a host file is being used (see the description of *hfile* in the “Parameters” section above), the default data formats are as follows:
  - The order, type, length, and number of the columns in the host file are assumed to be identical to the order, type, and number of the columns in the database table.
  - If a given database column’s data is fixed-length, then the host file’s data column will also be fixed-length. If a given database column’s data is variable-length or may contain null values, the host file’s data column will be prefixed by a 4-byte length value for SYBTEXT and SYBIMAGE data types, and a 1-byte length value for all other types.
  - There are no terminators of any kind between host file columns.

Any of these defaults can be overridden by calling `bcp_columns` and `bcp_colfmt`.

- Using the bulk copy routines to copy data to a database table requires the following:
  - The DBPROCESS structure must be usable for bulk copy purposes. This is accomplished by calling `BCP_SETL`:

```
login = dblogin();
BCP_SETL(login, TRUE);
```

- If the table has no indexes, the database option `select into/bulkcopy` must be set to “true.” The following SQL command will do this:

```
sp_dboption 'mydb', 'select into/bulkcopy',
'true'
```

See the *Adaptive Server Enterprise Reference Manual* for further details on `sp_dboption`.

- If no host file is being used, it is necessary to call `bcp_bind` to specify the format and location in memory of each column’s data value.

- If no host file is being used, *errfile* is ignored. An error file is not necessary when bulk-copying from program variables because bcp\_sendrow returns FAIL if an error occurs. In this case, the application can examine the bulk copy program variables to determine which row values caused the error.

See also

bcp\_batch, bcp\_bind, bcp\_colfmt, bcp\_collen, bcp\_colptr, bcp\_columns, bcp\_control, bcp\_done, bcp\_exec, bcp\_sendrow

## bcp\_moretext

**Description** Send part of a text or image value to Adaptive Server Enterprise.

**Syntax** RETCODE bcp\_moretext(dbproc, size, text)

DBPROCESS \*dbproc;  
DBINT size;  
BYTE \*text;

**Parameters**

**dbproc**

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and Adaptive Server Enterprise.

**size**

The size of this particular part of the text or image value being sent to Adaptive Server Enterprise. It is an error to send more text or image bytes to Adaptive Server Enterprise than were specified in the call to bcp\_bind or bcp\_collen.

**text**

A pointer to the text or image portion to be written.

**Return value** SUCCEED or FAIL.

**Usage**

- This routine is used in conjunction with bcp\_bind and bcp\_sendrow to send a large SYBTEXT or SYBIMAGE value to Adaptive Server Enterprise in the form of a number of smaller chunks. This is particularly useful with operating systems unable to allocate extremely long data buffers.



- If `bcp_bind` is called with a *type* parameter of `SYBTEXT` or `SYBIMAGE` and a non-NULL *varaddr* parameter, `bcp_sendrow` will send the entire text or image data value, just as it does for all other datatypes. If, however, `bcp_bind` has a NULL *varaddr* parameter, `bcp_sendrow` will return control to the application immediately after all non-text or image columns are sent to Adaptive Server Enterprise. The application can then call `bcp_moretext` repeatedly to send the text and image columns to Adaptive Server Enterprise, a chunk at a time.
- Here is an example that illustrates how to use `bcp_moretext` with `bcp_bind` and `bcp_sendrow`:

```

LOGINREC      *login;
DBPROCESS     *dbproc;

DBINT         id = 5;
char          *part1 = "This text value isn't very long,";
char          *part2 = " but it's broken up into three parts";
char          *part3 = " anyhow.";

/* Initialize DB-Library. */
if (dbinit() == FAIL)
    exit(ERREXIT);

/* Install error handler and message handler. */
dberrhandle(err_handler);
dbmsghandle(msg_handler);

/* Open a DBPROCESS */
login = dblogin();
BCP_SETL(login, TRUE);
dbproc = dbopen(login, NULL);

/* Initialize bcp. */
if (bcp_init(dbproc, "comdb..articles", (BYTE *)NULL,
            (BYTE *)NULL, DB_IN) == FAIL)
    exit(ERREXIT);

/* Bind program variables to table columns. */
if (bcp_bind(dbproc, (BYTE *)&id, 0, (DBINT)-1,
            (BYTE *)NULL, 0, SYBINT4, 1) == FAIL)
{
    fprintf(stderr, "bcp_bind, column 1, failed.\n");
    exit(ERREXIT);
}

```

```
if (bcp_bind
    (dbproc, (BYTE *)NULL, 0,
     (DBINT) (strlen(part1) + strlen(part2) + strlen(part3)),
     (BYTE *)NULL, 0, SYBTEXT, 2)
    == FAIL)
{
    fprintf(stderr, "bcp_bind, column 2, failed.\n");
    exit(ERREXIT);
}

/*
** Now send this row, with the text value broken into
** three chunks.
*/
if (bcp_sendrow(dbproc) == FAIL)
    exit(ERREXIT);
if (bcp_moretext(dbproc, (DBINT)strlen(part1), part1) == FAIL)
    exit(ERREXIT);
if (bcp_moretext(dbproc, (DBINT)strlen(part2), part2) == FAIL)
    exit(ERREXIT);
if (bcp_moretext(dbproc, (DBINT)strlen(part3), part3) == FAIL)
    exit(ERREXIT);

/* We're all done. */
bcp_done(dbproc);
dbclose(dbproc);
```

- If you use `bcp_moretext` to send one text or image column in the row, you must also use it to send all other text and image columns in the row.
- If the row contains more than one text or image column, `bcp_moretext` will first send its data to the lowest-numbered (that is, leftmost) text or image column, followed by the next lowest-numbered column, and so on.
- An application will normally call `bcp_sendrow` and `bcp_moretext` within loops, to send a number of rows of data. Here is an outline of how to do this for a table containing two text columns:

```
while (there are still rows to send)
{
    bcp_sendrow(...);

    for (all the data in the first text column)
        bcp_moretext(...);

    for (all the data in the second text column)
        bcp_moretext(...);
}
```

}

See also `bcp_bind`, `bcp_sendrow`, `dbmoretext`, `dbwritetext`

## bcp\_options

Description Set bulk copy options.

Syntax `RETCODE bcp_options (dbproc, option, value, valuelen)`

`DBPROCESS *dbproc;`  
`BYTE *value;`  
`int valuelen;`

Parameters `dbproc`  
A pointer to the `DBPROCESS` structure that provides the connection for a particular front-end/Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and Adaptive Server Enterprise.

`value`  
A generic `BYTE` pointer to the value of the specified option. As the following table describes, what value should point to depends on *option*:

**Table 3-1: Values for value (bcp\_options)**

| If option is | *value should be  |
|--------------|---|
| BCPLABELED   | A <code>DBBOOL</code> value. Set <i>*value</i> to “true” to allow a bulk copy with sensitivity labels. Set <i>*value</i> to “false” for a normal bulk copy operation. |

`valuelen`  
The length of the data to which *value* points. If *value* points to a fixed-length item (for example a `DBBOOL`, `DBINT`, and so on), pass *valuelen* as -1.

Return value `SUCCEED` or `FAIL`.

Usage

- `bcp_options` sets bulk copy options.
- Currently the only bulk copy option available is `BCPLABELED`.

See also `bcp_init`, `bcp_control`

## bcp\_readfmt

|              |   |
|--------------|---|
| Description  | Read a datafile format definition from a host file.   |
| Syntax       | <pre>RETCODE bcp_readfmt(dbproc, filename)</pre><br><pre>DBPROCESS *dbproc;<br/>char      *filename;</pre>  |
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front end/Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and Adaptive Server Enterprise.</p> <p><b>filename</b><br/>The full directory specification of the file containing the format definitions.</p>   |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"><li>• bcp_readfmt reads a datafile format definition from a host file, then makes the appropriate calls to bcp_columns and bcp_colfmt. This automates the bulk copy of multiple files that share a common data format.</li><li>• bcp, the bulk copy utility, copies a database table to or from a host file in a user-specified format. User-specified formats may be saved through bcp in datafile format definition files, which can later be used to automate the bulk copy of files that share a common format. See the <i>Open Client and Open Server Programmers Supplement</i>.</li><li>• Application programs can call bcp_writfmt to create files with datafile format definitions.</li><li>• The following code fragment illustrates the use of bcp_readfmt:<pre>bcp_init(dbproc, "mytable", "bcpdata", "bcperes", DB_IN);<br/>bcp_readfmt(dbproc, "my_fmtfile");<br/>bcp_exec(dbproc, &amp;rows_copied);</pre></li></ul> |
| See also     | bcp_colfmt, bcp_columns, bcp_writfmt  |

## bcp\_sendrow

|             |  |
|-------------|--|
| Description | Send a row of data from program variables to Adaptive Server Enterprise. |
|-------------|--|

|              |  |
|--------------|--|
| Syntax       | <pre>RETCODE bcp_sendrow(dbproc)  DBPROCESS *dbproc;</pre>   |
| Parameters   | <p><b>dbproc</b></p> <p>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and Adaptive Server Enterprise.</p>  |
| Return value | SUCCEED or FAIL.   |
| Usage        | <ul style="list-style-type: none"> <li>• <code>bcp_sendrow</code> builds a row from program variables and sends it to Adaptive Server Enterprise.</li> <li>• Before calling <code>bcp_sendrow</code>, you must make calls to <code>bcp_bind</code> to specify the program variables to be used.</li> <li>• If <code>bcp_bind</code> is called with a <i>type</i> parameter of SYBTEXT or SYBIMAGE and a non-null <i>varaddr</i> parameter, <code>bcp_sendrow</code> will send the entire text or image data value, just as it does for all other datatypes. If, however, <code>bcp_bind</code> has a null <i>varaddr</i> parameter, <code>bcp_sendrow</code> will return control to the application immediately after all non-text or image columns are sent to Adaptive Server Enterprise. The application can then call <code>bcp_moretext</code> repeatedly to send the text and image columns to Adaptive Server Enterprise, a chunk at a time. For an example, see the <code>bcp_moretext</code> reference page.</li> <li>• After the last call to <code>bcp_sendrow</code>, you must call <code>bcp_done</code> to ensure proper internal cleanup.</li> <li>• When <code>bcp_sendrow</code> is used to bulk copy rows from program variables into Adaptive Server Enterprise tables, rows are permanently saved in Adaptive Server Enterprise only when the user calls <code>bcp_batch</code> or <code>bcp_done</code>.</li> </ul> <p>The user may choose to call <code>bcp_batch</code> once every <i>n</i> rows, or when there is a lull between periods of incoming data (as in a telemetry application). Of course, the user may choose some other criteria or may decide not to call <code>bcp_batch</code> at all. If <code>bcp_batch</code> is never called, the rows are permanently saved in Adaptive Server Enterprise when <code>bcp_done</code> is called.</p> |
| See also     | <code>bcp_batch</code> , <code>bcp_bind</code> , <code>bcp_colfmt</code> , <code>bcp_collcn</code> , <code>bcp_colptr</code> , <code>bcp_columns</code> , <code>bcp_control</code> , <code>bcp_done</code> , <code>bcp_exec</code> , <code>bcp_init</code> , <code>bcp_moretext</code>   |

## BCP\_SETL

|              |  |
|--------------|--|
| Description  | Set the LOGINREC for bulk copy operations into the database.   |
| Syntax       | RETCODE BCP_SETL(loginrec, enable)<br><br>LOGINREC    *loginrec;<br>DBBOOL     enable;   |
| Parameters   | loginrec<br>This is a pointer to a LOGINREC structure, which will be passed as an argument to dbopen. You can get a LOGINREC structure by calling dblogin.<br><br>enable<br>This is a Boolean value (“true” or “false”) that specifies whether or not to enable bulk copy operations for the resulting DBPROCESS. By default, DBPROCESSes are <i>not</i> enabled for bulk copy operations.   |
| Return value | SUCCEED or FAIL.   |
| Usage        | <ul style="list-style-type: none"><li>• This macro sets a field in the LOGINREC structure that tells Adaptive Server Enterprise that the DBPROCESS connection may be used for bulk copy operations. To have any effect, it must be called before dbopen, the routine that actually allocates the DBPROCESS structure.</li><li>• Applications that allow users to make ad hoc queries may want to avoid calling BCP_SETL (or call it with a value of “false” for the <i>enable</i> parameter) to prevent users from initiating a bulk copy sequence through SQL commands. Once a bulk copy sequence has begun, it cannot be stopped through an ordinary SQL command.</li><li>• BCP_SETL applies to “copy in” operations only.</li></ul> |
| See also     | bcp_init, bcp_getl, dblogin, dbopen, DBSETLAPP, DBSETLHOST, DBSETLPWD, DBSETLUSER  |

## bcp\_setxlate

|             |   |
|-------------|---|
| Description | Specify the character set translations to use when retrieving data from or inserting data into an Adaptive Server Enterprise.             |
| Syntax      | RETCODE bcp_setxlate(dbproc, xlt_tosrv, xlt_todisp)<br><br>DBPROCESS    *dbproc;<br>DBXLATE      *xlt_tosrv;<br>DBXLATE      *xlt_todisp; |

|              |   |
|--------------|---|
| Parameters   | <p><b>dbproc</b><br/>A pointer to the DBPROCESS structure that provides the connection for a particular front end/Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and Adaptive Server Enterprise.</p> <p><b>xlt_tosrv</b><br/>A pointer to a translation structure. The translation structure is allocated using <code>dbload_xlate</code>. <i>xlt_tosrv</i> indicates the character set translation to use when moving data from the application program to the Adaptive Server Enterprise (the copy <i>in</i>, or DB_IN, direction).</p> <p><b>xlt_todisp</b><br/>A pointer to a translation structure. The translation structure is allocated using <code>dbload_xlate</code>. <i>xlt_todisp</i> indicates the character set translation to use when moving data from Adaptive Server Enterprise to the application program (the copy <i>out</i>, or DB_OUT, direction).</p>  |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"> <li>• <code>bcp_setxlate</code> specifies the character set translations to use when transferring character data between the Adaptive Server Enterprise and a front-end application program using <code>bcp</code>.</li> <li>• The specified character set translations need not be the same as those being used to display or input data on the user's terminal. The translations may be used to read or write a data file in a completely different character set that is not intended for immediate display.</li> <li>• The following code fragment illustrates the use of <code>bcp_setxlate</code>: <pre> bcp_init(dbproc, "mytable", "bcpdata", "bcperres", DB_OUT); bcp_setxlate(dbproc, xlt_tosrv, xlt_todisp); bcp_columns(dbproc, 3); bcp_colfmt(dbproc, 1, SYBCHAR, 0, -1, "\t", 1, 1); bcp_colfmt(dbproc, 2, SYBCHAR, 0, -1, "\t", 1, 2); bcp_colfmt(dbproc, 3, SYBCHAR, 0, -1, "\n", 1, 3); bcp_exec(dbproc); </pre> </li> </ul> <p>See also <code>dbfree_xlate</code>, <code>dbload_xlate</code>, <code>dbxlate</code></p> |

## bcp\_writfmt

**Description** Write a datafile format definition to a host file.

|              |   |
|--------------|---|
| Syntax       | RETCODE bcp_writfmt(dbproc, filename)<br><br>DBPROCESS *dbproc;<br>char *filename;  |
| Parameters   | <p>dbproc<br/>A pointer to the DBPROCESS structure that provides the connection for a particular front-end/Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and Adaptive Server Enterprise.</p> <p>filename<br/>The full directory specification of the file that contains the format definitions.</p>  |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"><li>• bcp_writfmt writes a datafile format definition to a host file. The format reflects previous calls to bcp_columns and bcp_colfmt.</li><li>• bcp, the bulk copy utility, copies a database table to or from a host file in a user-specified format. User-specified formats may be saved through bcp in “datafile format definition files,” which can later be used to automate the bulk copy of files that share a common format. See the <i>Open Client and Open Server Programmers Supplement</i>.</li><li>• Format definition files are read using bcp_readfmt.</li><li>• The following code fragment illustrates the use of bcp_writfmt:</li></ul> <pre>bcp_init(dbproc, "mytable", "bcpdata", "bcperrs", DB_OUT);<br/><br/>bcp_columns(dbproc, 3);<br/>bcp_colfmt(dbproc, 1, SYBCHAR, 0, -1, "\t", 1, 1);<br/>bcp_colfmt(dbproc, 2, SYBCHAR, 0, -1, "\t", 1, 2);<br/>bcp_colfmt(dbproc, 3, SYBCHAR, 0, -1, "\n", 1, 3);<br/><br/>bcp_writfmt(dbproc, "my_fmtfile");<br/>bcp_exec(dbproc, &amp;rows_copied);</pre> |
| See also     | bcp_colfmt, bcp_columns, bcp_readfmt  |



## Two-Phase Commit Service

Adaptive Server Enterprise provides a two-phase commit service that allows a client application to coordinate transactions that are distributed on two or more Adaptive Server Enterprises.

This chapter describes the two-phase commit process and the DB-Library routines that are involved.

| Topic  | Page |
|--|------|
| Programming distributed transactions           | 453  |
| The commit service and the application program | 454  |
| The probe process                              | 456  |
| Two-phase commit routines                      | 456  |
| Specifying the commit server                   | 457  |
| Two-phase commit sample program                | 458  |
| Program notes                                  | 464  |

### Programming distributed transactions

The two-phase commit service allows an application to coordinate updates among two or more Adaptive Server Enterprises. This initial implementation of distributed transactions treats separate transactions (which may be on separate Adaptive Server Enterprises) as if they were a single transaction. The commit service uses one Adaptive Server Enterprise, the “commit server,” as a central record-keeper that helps the application determine whether to commit, or whether to roll back transactions in case of failure. Thus, the two-phase commit guarantees that either all or none of the databases on the participating servers are updated.

A distributed transaction is performed by submitting Transact-SQL statements to the Adaptive Server Enterprises through DB-Library routines. An application program opens a session with each server, issues the update commands, and then prepares to commit the transaction. Through DB-Library, the application issues the following to each participating server:

- A begin transaction with identifying information on the application, the transaction, and the commit server
- The Transact-SQL update statements
- A prepare transaction statement that indicates that the updates have been performed and that the server is prepared to commit

After the updates have been performed on all the servers participating in the distributed transaction, the two-phase commit begins. In the first phase, all servers agree that they are ready to commit. In the second phase, the application informs the commit service that the transaction is complete (that is, the commit will take place), and a commit transaction is then issued to all of the servers, causing them to commit.

If an error occurs between phase one and phase two, all servers coordinate with the commit service to determine whether the transaction should be committed or aborted.

---

**Note** If certain types of errors occur during a two-phase transaction, Adaptive Server Enterprise may need to mark a two-phase process as “infected.” Marking the process as infected rather than killing it aids in later error recovery. To ensure that Adaptive Server Enterprise is able to mark processes as infected, boot Adaptive Server Enterprise with the flag `-T3620` passed on the command line.

---

## The commit service and the application program

The role of the commit service is to be a single place of record that helps the application decide whether the transaction should be committed or aborted.

If the Adaptive Server Enterprises are all prepared to commit, the application notifies the commit service to mark the transaction as committed. Once this happens, the transaction is committed despite any failures that might subsequently happen.

If any Adaptive Server Enterprise or the application program fails before the prepare transaction statement, the Adaptive Server Enterprise will rollback the transaction.

If any Adaptive Server Enterprise or the application program fails after the prepare but before the commit, the Adaptive Server Enterprise will communicate with the server functioning as the commit service and ask it whether to rollback or commit.

If the Adaptive Server Enterprise cannot communicate with the server functioning as the commit service, it will mark the user task process as infected in Adaptive Server Enterprise. At this point, the System Administrator can either kill the infected process immediately, or wait until communication to the commit service is restored to kill the infected process.

- If the System Administrator kills the infected process immediately, two-phase commit protocol is violated and the integrity of the two-phase transaction is not guaranteed. Servers participating in the transaction may be in inconsistent states.
- If the System Administrator kills the infected process after communication with the commit service has been restored, the Adaptive Server Enterprise will communicate with the commit service to determine whether or not to commit the transaction locally. The integrity of the two-phase transaction is guaranteed.

To decide whether or not to kill the infected process immediately, the System Administrator must consider the estimated downtime of the commit service, the number and importance of locks held by the infected process, and the complexity of the transaction in progress.

The role of the application program is to deliver the Transact-SQL statements to the Adaptive Server Enterprises in the proper order, using the proper DB-Library routines. The role of the commit service is to provide a single place where the commit/rollback status is maintained. The Adaptive Server Enterprises communicate with the commit service only if a failure happens during the two-phase commit.

The commit service needs its own DBPROCESS, separate from the DBPROCESSes used for the distributed transaction, to perform its record-keeping. Note, however, that the server handling the commit service can also be one of the servers participating in the transaction, as long as the commit service has its own DBPROCESS. In fact, all the servers involved in the transaction can be one and the same.

## The probe process

If any server must recover the transaction, it initiates a process, probe, that determines the last known status of the transaction. After it returns the status of that transaction to the commit service, the probe process dies. The probe process makes use of `stat_xact`, the same status-checking routine that the commit service uses to check the progress of a distributed transaction.

## Two-phase commit routines

The following routines make up the two-phase commit service:

| Routine                        | Description  |
|--------------------------------|--|
| <code>abort_xact</code>        | Tells the commit service to abort the transaction.   |
| <code>build_xact_string</code> | Builds a name string for use by each participating Adaptive Server Enterprise for its begin transaction and prepare transaction statements. This string encodes the application's transaction name, the commit service name, and the <i>commid</i> .   |
| <code>close_commit</code>      | Closes the connection with the commit service.   |
| <code>commit_xact</code>       | Tells the commit service to commit the transaction.  |
| <code>open_commit</code>       | Opens a connection with the commit service. The routine is given the login ID of the user initiating the session and the name of the commit service. It returns a pointer to a <code>DBPROCESS</code> structure used in subsequent commit service calls.   |
| <code>remove_xact</code>       | Decrements the count of servers still participating in the transaction.  |
| <code>start_xact</code>        | Records the start of a distributed transaction and stores initial information about the transaction ( <code>DBPROCESS</code> id, application name, transaction name, and number of sites participating) in a lookup table on the commit server. It returns the <i>commid</i> identifying number for the transaction. |

Two additional routines are used for ongoing status reports:

| Routine                | Description   |
|------------------------|---|
| <code>scan_xact</code> | Returns the status of a single transaction or all distributed transactions. |
| <code>stat_xact</code> | Returns the completion status of a distributed transaction.                 |

During the course of a session, the diagnostic routines `scan_xact` and `stat_xact` are used to check that the commit service carried out the request.

The `scan_xact` routine uses the commit service lookup table, `spt_committab`, which holds the following values:

- Transaction ID
- Time the transaction started
- Last time the row was updated
- Number of servers initially involved in the transaction
- Number of servers that have not yet completed
- Status: “a” (abort), “c” (commit), “b” (begin)
- Application name
- Transaction name

The two-phase commit routines call internal stored procedures (for example, `sp_start_xact`) that are created in each server’s master database. The *installmaster* script creates the commit service lookup table and stored procedures in each server’s master database, for use whenever that server becomes a commit server.

## Specifying the commit server

The commit server must have an entry in the `interfaces` file on each machine participating in the distributed transaction. On the machine on which the commit server is actually running, the commit server entry must specify the usual ports described in the *Open Client and Open Server Configuration Guide*, including a query port. For example:

```
SERVICE
master tcp sun-ether rose 2001
query tcp sun-ether rose 2001
```

On any additional machines containing other servers participating in the distributed transaction, the commit server entry need to specify only the query port:

```
SERVICE
query tcp sun-ether rose 2001
```

```
SITEA
master tcp sun-ether gaia 2011
query tcp sun-ether gaia 2011
```

The name of the commit server (in these examples, “SERVICE”) is used as a parameter in calls to the `open_commit` and `build_xact_string` routines. The commit server name must be the same on all machines participating in the transaction. The name cannot contain a period (.) or a colon (:).

## Two-phase commit sample program

An sample program illustrating the two-phase commit service is included with DB-Library’s sample programs. This same example is duplicated below, but with comments added to document how recovery occurs for the different types of failure that may occur at various points in the transaction.

```
/*
**      twophase.c
**
**      Demo of Two-Phase Commit Service
**
**      This example uses the two-phase commit service
**      to perform a simultaneous update on two servers.
**      In this example, one of the servers participating
**      in the distributed transaction also functions as
**      the commit service.
**
**      In this particular example, the same update is
**      performed on both servers. You can, however, use
**      the commit server to perform completely different
**      updates on each server.
**
**/

#include <stdio.h>
#include <sybfront.h>
#include <sybdb.h>
#include "sybdbex.h"

int err_handler();
int msg_handler();

char      cmdbuf[256];
```

```
char      xact_string[128];

main()
{

    DBPROCESS      *dbproc_server1;
    DBPROCESS      *dbproc_server2;
    DBPROCESS      *dbproc_commit;
    LOGINREC       *login;
    int            commid;

    RETCODE         ret_server1;
    RETCODE         ret_server2;

    /* Initialize DB-Library. */
    if (dbinit() == FAIL)
        exit (ERREXIT);

    dberrhandle(err_handler);
    dbmsghandle(msg_handler);

    printf("Demo of Two Phase Commit\n");

    /* Open connections with the servers and the
    ** commit service. */
    login = dblogin();
    DBSETLPWD(login, "server_password");
    DBSETLAPP(login, "twophase");

    dbproc_server1 = dbopen (login, "SERVICE");
    dbproc_server2 = dbopen (login, "PRACTICE");
    dbproc_commit = open_commit (login, "SERVICE");

    if (dbproc_server1 == NULL ||
        dbproc_server2 == NULL ||
        dbproc_commit == NULL)
    {
        printf (" Connections failed!\n");
        exit (ERREXIT);
    }

    /* Use the "pubs2" database. */
    sprintf(cmdbuf, "use pubs2");
    dbcmd(dbproc_server1, cmdbuf);
    dbsqlexec(dbproc_server1);
```

```
dbcmd(dbproc_server2, cmdbuf);
dbsqllexec(dbproc_server2);

/*
** Start the distributed transaction on the
** commit service.
*/
commid = start_xact(dbproc_commit, "demo", "test", 2);
```

---

**Note** The application is now in the *begin* phase of the two-phase commit transaction.

---

```
/* Build the transaction name. */
build_xact_string("test", "SERVICE", commid, xact_string);

/* Build the first command buffer. */
sprintf(cmdbuf, "begin transaction %s", xact_string);

/* Begin the transactions on the different servers. */
dbcmd(dbproc_server1, cmdbuf);
dbsqllexec(dbproc_server1);
dbcmd(dbproc_server2, cmdbuf);
dbsqllexec(dbproc_server2);

/* Do various updates. */
sprintf(cmdbuf, " update titles set price = $1.50 where");
strcat(cmdbuf, " title_id = 'BU1032'");
dbcmd(dbproc_server1, cmdbuf);
ret_server1 = dbsqllexec(dbproc_server1);
dbcmd(dbproc_server2, cmdbuf);
ret_server2 = dbsqllexec(dbproc_server2);
```

---

**Note** See “Program note 1” on page 464.

---

```
if (ret_server1 == FAIL || ret_server2 == FAIL)
{
    /* Some part of the transaction failed. */
    printf(" Transaction aborted -- dbsqllexec failed\n");
    abortall(dbproc_server1, dbproc_server2,
             dbproc_commit, commid);
}

/* Find out if all servers can commit the transaction. */
sprintf(cmdbuf, "prepare transaction");
```



```
dbcmd(dbproc_server1, cmdbuf);
dbcmd(dbproc_server2, cmdbuf);
ret_server1 = dbsqlexec(dbproc_server1);
```

---

**Note** See “Program note 2” on page 464.

---

```
ret_server2 = dbsqlexec(dbproc_server2);
```

---

**Note** See “Program note 3” on page 465.

---

```
if (ret_server1 == FAIL || ret_server2 == FAIL)
{
    /* One or both of the servers failed to prepare. */
    printf(" Transaction aborted -- prepare failed\n");
    abortall(dbproc_server1, dbproc_server2,
            dbproc_commit, commid);
}
```

---

**Note** See “Program note 4” on page 465.

---

```
/* Commit the transaction. */
if (commit_xact(dbproc_commit, commid) == FAIL)
{
    /* The commit server failed to record the commit. */
    printf(" Transaction aborted -- commit_xact failed\n");
    abortall(dbproc_server1, dbproc_server2,
            dbproc_commit, commid);
    exit(ERREXIT);
}
```

---

**Note** See “Program note 5” on page 466.

---

```
/* The transaction has successfully committed.
** Inform the servers.
*/
sprintf(cmdbuf, "commit transaction");
dbcmd(dbproc_server1, cmdbuf);
if (dbsqlexec(dbproc_server1) != FAIL)
    remove_xact(dbproc_commit, commid, 1);
```

---

**Note** See “Program note 6” on page 466.

---

```
dbcmd(dbproc_server2, cmdbuf);
if (dbsqlxec(dbproc_server2) != FAIL)
    remove_xact(dbproc_commit, commid, 1);
```

---

**Note** See “Program note 7” on page 467.

---

```
/* Close the connection to the commit server. */
close_commit(dbproc_commit);
```

---

**Note** See “Program note 8” on page 467.

---

```
printf( "We made it!\n");
dbexit();
exit(STDEXIT);
}

/* Function to abort the distributed transaction. */

abortall( dbproc_server1, dbproc_server2, dbproc_commit, commid )
DBPROCESS      *dbproc_server1;
DBPROCESS      *dbproc_server2;
DBPROCESS      *dbproc_commit;
int            commid;
{
    /* Some part of the transaction failed. */

    /* Inform the commit server of the failure. */
    abort_xact(dbproc_commit, commid);

    /* Roll back the transactions on the different servers. */
    sprintf(cmdbuf, "rollback transaction");
    dbcmd(dbproc_server1, cmdbuf);
    if (dbsqlxec(dbproc_server1) != FAIL)
        remove_xact(dbproc_commit, commid, 1);
    dbcmd(dbproc_server2, cmdbuf);
    if (dbsqlxec(dbproc_server2) != FAIL)
        remove_xact(dbproc_commit, commid, 1);

    dbexit();
    exit(ERREXIT);
}

/* Message and error handling functions. */
```

```
int msg_handler(dbproc, msgno, msgstate, severity, msgtext,
               servername, procname, line)

DBPROCESS      *dbproc;
DBINT          msgno;
int            msgstate;
int            severity;
char           *msgtext;
char           *servername;
char           *procname;
DBUSMALLINT    line;

{
    /* Msg 5701 is just a use database message, so skip it. */
    if (msgno == 5701)
        return (0);

    /* Print any severity 0 message as is, without extra stuff. */
    if (severity == 0)
    {
        (void) fprintf (ERR_CH, "%s\n", msgtext);
        return (0);
    }

    (void) fprintf (ERR_CH, "Msg %ld, Level %d, State %d\n",
                    msgno, severity, msgstate);

    if (strlen(servername) > 0)
        (void) fprintf (ERR_CH, "Server '%s', ", servername);
    if (strlen(procname) > 0)
        (void) fprintf (ERR_CH, "Procedure '%s', ", procname);
    if (line > 0)
        (void) fprintf (ERR_CH, "Line %d", line);

    (void) fprintf (ERR_CH, "\n\t%s\n", msgtext);

    if (severity >= 16)
    {
        (void) fprintf (ERR_CH, "Program Terminated! Fatal\
Adaptive Server Enterprise error.\n");
        exit(ERREXIT);
    }

    return (0);
}
```

```
int err_handler(dbproc, severity, dberr, oserr, dberrstr, oserrstr)
DBPROCESS      *dbproc;
int             severity;
int             dberr;
int             oserr;
char            *dberrstr;
char            *oserrstr;
{
    if ((dbproc == NULL) || (DBDEAD(dbproc)))
        return (INT_EXIT);
    else
    {
        (void) fprintf (ERR_CH, "DB-Library error: \
                               \n\t %s\n", dberrstr);
        if (oserr != DBNOERR)
            (void) fprintf (ERR_CH, "Operating system error:\
                               \n\t%s\n", oserrstr);
    }

    return (INT_CANCEL);
}
```

## Program notes

This section contains the notes referenced in the sample code.

### Program note 1

If any type of failure occurs at this point, it is the application's responsibility to roll back the transactions using `abort_xact`.

### Program note 2

The application has entered the *prepare* stage of the two-phase commit transaction. As far as the commit server is aware, however, the application is still in the *begin* phase.

### Program note 3

If any type of failure occurs at this point, it is the application's responsibility to roll back the transactions using `abort_xact`.

### Program note 4

At this point, the following failures are possible:

- The application's link to the commit server, or the commit server itself, may go down.

In this case, the following call to `commit_xact` fails, and the application must roll back the transactions using `abort_xact`.

- The application's link to a participating server may go down.

In this case, the following call to `commit_xact` will succeed, but the application's commit transaction command to the participating server will not. However, the server will be aware that its connection with the application has died. It will communicate with the commit server, using `probe`, to determine whether to commit the transaction locally.

- A participating server may go down.

In this case, the following call to `commit_xact` will succeed, but the application's commit transaction to the participating server will not. When the participating server comes back up, it will use `probe` to determine whether to commit the transaction locally.

- Both the application's link to the commit server and the application's link to the participating server may go down.

In this case, the following call to `commit_xact` fails. The application must roll back the transactions with `abort_xact`, but will not be able to communicate with the participating server. The participating server will use `probe` to communicate with the commit server. It will learn that the transaction has not been committed in the commit service, and will roll back the transaction locally.

- Both the application's link to the participating server and the participating server's link to the commit server may go down.

In this case, the following call to `commit_xact` will succeed, but the application will not be able to communicate this to the participating server. When its connection to the application dies, the participating server will attempt to communicate with the commit server using probe to determine whether or not to commit the transaction locally. Because its link to the commit server is down, however, it will not be able to.

Because it cannot resolve the transaction, the participating server marks the user task process as infected.

If the System Administrator kills the infected process while the commit server is still down, two-phase commit protocol is violated and the integrity of the transaction is not guaranteed.

If the System Administrator waits until commit server is back up to kill the infected process, probe executes automatically when the System Administrator attempts to kill the process. probe communicates with the commit server and determines whether the participating server should commit the transaction locally. The integrity of the transaction is guaranteed.

## Program note 5

The application has entered the *committed* phase of the two-phase commit transaction. This means that any probe process querying the commit server will be told to commit the transaction locally. After this point, the application does not need to concern itself with aborting the transaction.

## Program note 6

If the above `dbsqlxexec` to Server1 fails because the application's link to the server has gone down, Server1 will use probe to communicate with the commit server. probe will find that the transaction is committed in the commit server and will tell Server1 to commit locally.

If probe cannot communicate with the commit server, Server1 will infect the user task process in Adaptive Server Enterprise. If the System Administrator kills the infected process before communication with the commit server is reestablished, the transaction will be rolled back, thus violating two-phase protocol and leaving the database in an inconsistent state. If possible, the System Administrator should always wait until communication with the commit server is reestablished before killing the infected process.

If the `dbsqlxexec` to Server1 fails because Server1 has gone down, the local transaction will remain in a suspended state until Server1 is restored. As part of the recovery process, Server1 will use probe to communicate with the commit server. probe will find that the transaction is committed in the commit server and will tell Server1 to commit locally.

If probe cannot communicate with the commit server, Server1 will mark the database as suspect. After communication with the commit sever is reestablished, the suspect database should be re-recovered.

## Program note 7

If the above `dbsqlxexec` to Server2 fails because the application's link to the server has gone down, Server2 will use probe to communicate with the commit server. probe will find that the transaction is committed in the commit server and will tell Server2 to commit locally.

If probe cannot communicate with the commit server, Server2 will infect the user task process in Adaptive Server Enterprise. If the System Administrator kills the infected process before communication with the commit server is reestablished, the transaction will be rolled back, thus violating two-phase protocol and leaving the database in an inconsistent state. If possible, the System Administrator should always wait until communication with the commit server is reestablished before killing the infected process.

If the `dbsqlxexec` to Server2 fails because Server2 has gone down, the local transaction will remain in a suspended state until Server2 is restored. As part of the recovery process, Server2 will use probe to communicate with the commit server. probe will find that the transaction is committed in the commit server and will tell Server2 to commit locally.

If probe cannot communicate with the commit server, Server2 will mark the database as suspect. After communication with the commit sever is reestablished, the suspect database should be re-recovered.

## Program note 8

`close_commit` marks the transaction as complete in the `spt_committab` table on the commit server. If `close_commit` fails, the transaction is not marked as complete. No actual harm is done by this, but the System Administrator may choose to manually update `spt_committab` in this case.

## abort\_xact

|              |  |
|--------------|--|
| Description  | Mark a distributed transaction as being aborted.   |
| Syntax       | <pre>RETCODE abort_xact(connect, commid)  DBPROCESS  *connect; DBINT      commid;</pre>  |
| Parameters   | <p><b>connect</b><br/>A pointer to the DBPROCESS used to communicate with the commit service.</p> <p><b>commid</b><br/>The <i>commid</i> used to identify the transaction to the commit service.</p> |
| Return value | SUCCEED or FAIL.   |
| Usage        | This routine informs the commit service that the status of a distributed transaction should be changed from “begin” to “abort.”  |
| See also     | commit_xact, remove_xact, scan_xact, start_xact, stat_xact   |

## build\_xact\_string

|             |  |
|-------------|--|
| Description | Build a name for a distributed transaction.  |
| Syntax      | <pre>void build_xact_string(xact_name, service_name,                       commid, result)  char    *xact_name; char    *service_name; DBINT   commid; char    *result;</pre>  |
| Parameters  | <p><b>xact_name</b><br/>The application or user name for the transaction. This name gets encoded in the name string but is not used by the commit service or Adaptive Server Enterprise. It serves to identify the transaction for debugging purposes.</p> |



**service\_name**

The name that will be used by Adaptive Server Enterprise to contact the commit service, should it be necessary to recover the transaction. If *service\_name* is NULL, the name DSCOMMIT is used.

*service\_name* must correspond to name of the interfaces file entry for the commit service. If *service\_name* is NULL, the interfaces file must contain an entry for DSCOMMIT.

**commid**

The number used by the commit service to identify the transaction. *commid* is the number returned by the call to *start\_xact*.

**result**

Address of buffer where the string should be built. The space must be allocated by the caller.

**Return value**

None.

**Usage**

- This routine builds a name string for use in the SQL begin transaction and prepare transaction of an Adaptive Server Enterprise transaction. If Adaptive Server Enterprise has to recover the transaction, it uses information encoded in the name to determine which commit service to contact and which transaction in that service to inquire about. The application should issue a SQL begin transaction using the string built by *build\_xact\_string*.
- The string built by *build\_xact\_string* must be large enough to hold the ASCII representation of *commid*, *xact\_name*, *service\_name*, two additional characters, and a null terminator.

**See also**

*commit\_xact*, *start\_xact*

## close\_commit

**Description**

End a connection with the commit service.

**Syntax**

```
void close_commit(connect)
```

```
DBPROCESS *connect;
```

**Parameters**

*connect*

A pointer to the DBPROCESS structure that was originally returned by *open\_commit*.

|              |   |
|--------------|---|
| Return value | None.   |
| Usage        | This routine calls <code>dbclose</code> to end a connection with the commit service. A call to <code>close_commit</code> should be made when the application is through with the commit service, to free resources. |
| See also     | <code>dbclose</code>  |

## **commit\_xact**

|              |  |
|--------------|--|
| Description  | Mark a distributed transaction as being committed.   |
| Syntax       | <code>RETCODE commit_xact(connect, commid)</code><br><br><code>DBPROCESS *connect;</code><br><code>DBINT commid;</code>  |
| Parameters   | <code>connect</code><br>A pointer to the <code>DBPROCESS</code> used to communicate with the commit service.<br><br><code>commid</code><br>The <i>commid</i> used to identify the transaction to the commit service. |
| Return value | <code>SUCCEED</code> or <code>FAIL</code> .<br><br>If <code>commit_xact</code> fails, you <i>must</i> roll back the transaction.   |
| Usage        | This routine informs the commit service that the status of a distributed transaction should be changed from “begin” to “commit.”   |
| See also     | <code>abort_xact</code> , <code>remove_xact</code> , <code>scan_xact</code> , <code>start_xact</code> , <code>stat_xact</code>   |

## **open\_commit**

|             |   |
|-------------|---|
| Description | Establish a connection with the commit service.   |
| Syntax      | <code>DBPROCESS *open_commit(login, servername)</code><br><br><code>LOGINREC *login;</code><br><code>char *servername;</code> |

|              |  |
|--------------|--|
| Parameters   | <p><b>login</b></p> <p>This is a LOGINREC containing information about the user initiating the session, such as login name, password, and options desired. The LOGINREC must have been obtained from a prior call to the DB-Library routine <code>dblogin</code>. The caller may wish to initialize fields in the LOGINREC. See the reference page for <code>dblogin</code> for more details.</p> <p><b>servername</b></p> <p>The name of the commit service; for example, <code>DSCOMMIT_SALESNET</code>. If <i>servername</i> is NULL, the name <code>DSCOMMIT</code> is used. The name cannot contain a period (.) or a colon (:).</p>  |
| Return value | A pointer to a DBPROCESS structure to be used in subsequent commit service calls, or NULL if the open failed.  |
| Usage        | <ul style="list-style-type: none"><li>• This routine calls <code>dbopen</code> to establish a connection with the commit service. A call to <code>open_commit</code> must precede any calls to other commit service routines, such as <code>start_xact</code>, <code>commit_xact</code>, <code>abort_xact</code>, <code>remove_xact</code>, and <code>scan_xact</code>. A session with the commit service is closed by calling <code>close_commit</code>.</li><li>• This routine returns a DBPROCESS structure, which is used to communicate with the commit service. The DBPROCESS must be dedicated to its role with the commit service and should not be used otherwise in the distributed transaction.</li></ul> |
| See also     | <code>dblogin</code> , <code>dbopen</code>   |

## remove\_xact

|             |  |
|-------------|--|
| Description | Decrement the count of sites still active in the distributed transaction.  |
| Syntax      | <pre>RETCODE remove_xact(connect, commid, n)</pre> <pre>DBPROCESS  *connect; DBINT      commid; int         n;</pre>   |
| Parameters  | <p><b>connect</b></p> <p>A pointer to the DBPROCESS used to communicate with the commit service.</p> <p><b>commid</b></p> <p>The <i>commid</i> used to identify the transaction to the commit service.</p> |

|              |   |
|--------------|---|
|              | n   |
|              | The number of sites to remove from the transaction.   |
| Return value | SUCCEED or FAIL.  |
| Usage        | <ul style="list-style-type: none"> <li>The commit service keeps a count of the number of sites participating in a distributed transaction. This routine informs the commit service that one or more sites has done a local commit or abort on the transaction and is hence no longer participating. The commit service removes the sites from the transaction by decrementing the count of sites.</li> <li>The transaction record is deleted entirely if the count drops to 0.</li> </ul> |
| See also     | abort_xact, commit_xact, scan_xact, start_xact, stat_xact   |

## scan\_xact

|              |  |
|--------------|--|
| Description  | Print commit service record for distributed transactions.  |
| Syntax       | RETCODE scan_xact(connect, commid)   |
|              | DBPROCESS   *connect;<br>DBINT        commid;  |
| Parameters   | <p>connect</p> <p>A pointer to the DBPROCESS used to communicate with the commit service.</p> <p>commid</p> <p>The <i>commid</i> used to identify the transaction to the commit service. If <i>commid</i> is -1, all commit service records are displayed.</p> |
| Return value | SUCCEED or FAIL.   |
| Usage        | This routine displays the commit service record for a specific distributed transaction, or for all distributed transactions known to the commit service.   |
| See also     | abort_xact, commit_xact, remove_xact, start_xact, stat_xact  |

## start\_xact

|             |   |
|-------------|---|
| Description | Start a distributed transaction using the commit service. |
|-------------|---|

|              |  |
|--------------|--|
| Syntax       | <pre>DBINT start_xact(connect, application_name, xact_name,                  site_count)  DBPROCESS *connect; char      *application_name; char      *xact_name; int       site_count;</pre>   |
| Parameters   | <p><b>connect</b><br/>A pointer to the DBPROCESS used to communicate with the commit service.</p> <p><b>application_name</b><br/>The name of the application. The application developer can choose any name for the application. It will appear in the table maintained by the commit service but is not used by the commit service or the Adaptive Server Enterprise recovery system.</p> <p><b>xact_name</b><br/>The name of the transaction. This name will appear in the table maintained by the commit service and must be supplied as part of the transaction name string built by <code>build_xact_string</code>. The name cannot contain a period (.) or a colon (:).</p> <p><b>site_count</b><br/>The number of sites participating in the transaction.</p> |
| Return value | An integer called the <i>commid</i> . This number is used to identify the transaction in subsequent calls to the commit service. In case of error, this routine will return 0.   |
| Usage        | This routine records the start of a distributed transaction with the commit service. A record is placed in the commit service containing the <i>commid</i> , which is a number that caller subsequently uses to identify the transaction to the commit service.  |
| See also     | <code>abort_xact</code> , <code>build_xact_string</code> , <code>commit_xact</code> , <code>remove_xact</code> , <code>scan_xact</code> , <code>stat_xact</code>   |

## stat\_xact

|             |   |
|-------------|---|
| Description | Return the current status of a distributed transaction. |
| Syntax      | <pre>int stat_xact(connect, commid)</pre>               |

|              |   |
|--------------|---|
|              | DBPROCESS    *connect;<br>DBINT        commid;  |
| Parameters   | <p>connect<br/>A pointer to the DBPROCESS used to communicate with the commit service.</p> <p>commid<br/>The <i>commid</i> is used to identify the transaction to the commit service. If <i>commid</i> is -1, all commit service records are displayed.</p> |
| Return value | A character code: “a” (abort), “b” (begin), “c” (commit), “u” (unknown), or -1 (request failed).  |
| Usage        | This routine returns the transaction status for the specified distributed transaction.  |
| See also     | abort_xact, commit_xact, remove_xact, scan_xact, start_xact   |

This appendix introduces the DB-Library cursor.

| Topic  | Page |
|--|------|
| Cursor overview                              | 475  |
| Sensitivity to change                        | 477  |
| DB-Library cursor functions                  | 480  |
| Holding locks                                | 480  |
| Stored procedures used by DB-Library cursors | 481  |

## Cursor overview

Because relational databases are oriented toward sets, no concept of next row exists, meaning that you cannot operate on an individual row in a set. Cursor functionality solves this problem by letting a result set be processed one row at a time, similar to the way you read and update a file on a disk. A DB-Library cursor indicates the current position in a result set, just as the cursor on your screen indicates the current position in a block of text.

DB-Library cursors are *client-side* cursors. This means that they do not correspond to an Adaptive Server Enterprise cursor, but emulate a cursor that appears to the user to be in the server. The DB-Library cursor transparently does keyset management, row positioning, and concurrency control entirely on the client side.

## DB-Library cursor capability

The DB-Library cursor routines offer the following capabilities, with certain limitations:

- Forward and backward scrolling (depending on how the keyset is defined during `dbcursoropen`)

- Direct access by position in the result set
- Positioned updates (even if the result set was defined with order by)
- Sensitivity adjustments to changes made by other users
- Concurrency control through several options

## Differences between DB-Library cursors and browse mode

Cursors let the user scroll through and update a result set with fewer restrictions than browse mode. Although cursors require a unique index, they do not require a timestamp nor a second connection to a database for updates. Also, they do not create a copy of the entire result set. The following table summarizes these differences:

**Table A-1: Cursors and browse mode**

| Item                             | Cursors             | Browse mode           |
|----------------------------------|---------------------|-----------------------|
| Row timestamps                   | Not required        | Required              |
| Multiple connections for updates | Unnecessary         | Necessary             |
| Table usage                      | Use original tables | Uses a copy of tables |

## Differences between DB-Library and Client-Library cursors

A DB-Library cursor does not correspond to an actual Adaptive Server Enterprise cursor. Instead, at the time the cursor is declared with `dbcursoropen`, DB-Library fetches keysets from Adaptive Server Enterprise “under the covers.” It then builds qualifiers based on the keys for the current row and sends them to Adaptive Server Enterprise. The server parses the query and returns a result set. When `dbcursorfetch` is called to retrieve more data, the DB-Library cursor may have to do additional selects. In addition, Adaptive Server Enterprise may have to parse the query each time `dbcursorfetch` is called.

A Client-Library cursor corresponds to an actual cursor in Adaptive Server Enterprise. It is sometimes referred to, therefore, as a *native* cursor. A new TDS protocol allows Client-Library to interact with the server to manage the cursor.

A Client-Library cursor is faster than a DB-Library cursor because it does not have to send SQL commands to the server, which causes multiple re-parsing of the query. But because the result set remains on the server side, it cannot offer the same options for concurrency control as a DB-Library cursor.



The following table summarizes these and additional differences between the two cursors:

**Table A-2: Differences between DB-Library cursors and Client-Library cursors**

| DB-Library cursor   | Client-Library cursor  |
|---|--|
| Cursor row position is defined by the client.   | Cursor row position is defined by the server.  |
| Can define optimistic concurrency control (allows dirty reads).   | Cannot define optimistic concurrency control (does not allow dirty reads).                         |
| Can fetch backward (if CUR_KEYSET or CUR_DYNAMIC is specified for <i>scrollopt</i> during <i>dbcursoropen</i> ).  | Can only fetch forward.  |
| More memory may be required if you query very large row sizes, unless you specify a smaller number of rows in the fetch buffer during <i>dbcursoropen</i> . | More memory is not required, regardless of how large the row sizes are.                            |
| You cannot access an Open Server application unless the application installs the required DB-Library stored procedures.                                     | You can access a version 10.0 (or later) Open Server application that is coded to support cursors. |
| Slower performance.   | Faster performance.  |

## Sensitivity to change

Three broad categories identify cursors according to their sensitivity to change:

- *Static* – values, order, and membership in the result set do not change while the cursor is open.
- *Keyset-driven* – values can change, but order and membership in the result set remain fixed at *open time* (the moment the cursor is opened).
- *Dynamic* – values, order, and membership in the result set can all change.

## Static cursor

In a static cursor, neither the cursor owner nor any other user can change the result set while the cursor is open. Values, membership, and order remain fixed until the cursor is closed. You can either take a snapshot of the result set (which begins to diverge from the snapshot as updates are made), or you can lock the entire result set to prevent updates.

It is not necessary for cursor routines to support static cursors directly. You can achieve static behavior through one of the following methods:

- Take a snapshot copy of the result set (with `select...into`), and then call `dbcursoropen` against the snapshot (temporary table).
- Lock the result set by calling `dbcursoropen` with the `holdlock` keyword in a `select` statement. However, this method significantly reduces concurrency.

## Keyset-driven cursor

In a keyset-driven cursor, the order and the membership of rows in the result set are fixed at open time, but changes to values may be made by the cursor owner. Committed changes made by other users are visible. If a change affects a row's order, or results in a row no longer qualifying for membership, the row does not disappear or move unless the cursor is closed and reopened. If the cursor remains open, deleted rows, when accessed, return a special error code that says they are missing. Updating the key also causes the rows to be "missing."

Inserted data does not appear, but changes to existing data do appear when the buffer is refreshed. With or without order by, the user can access rows by either *relative* or *absolute position*.

To access a row by relative position, move the cursor relative to its current position. For example, if the cursor is on row three and you want to access row eight, tell the cursor to jump five rows relative to its current position. The cursor jumps five rows to row eight.

To access a row by absolute position, tell the cursor the number of the row you want to access. For example, if the cursor is on row three and you want to access row eight, tell the cursor to jump to row eight.

## Dynamic cursor

In a dynamic cursor, uncommitted changes made by the cursor owner and committed changes made by anyone become visible the next time the user scrolls. Changes include inserts and deletes as well as changes in order and membership. (Deleted rows do not leave holes.) The user can access rows by relative (but not absolute) position in the result set. Dynamic cursors cannot use an order by clause.

## Concurrency control

Cursors control—through several options—*concurrent access*, which occurs when more than one user accesses and updates the same data at the same time. During concurrent access, data can become unreliable without some kind of control. To activate the particular concurrency control desired, specify one of the following options when you open a cursor:

**Table A-3: Concurrency control options**

| Option                      | Result  |
|-----------------------------|---|
| CUR_READONLY                | Updates are not permitted.  |
| CUR_LOCKCC                  | <p>The set of rows currently in the client buffer is locked when they are fetched inside a user-initiated transaction. No other user can update or read these rows. Updates issued by the cursor owner are guaranteed to succeed.</p> <p>No locks are held unless the application first issues begin transaction. Locks are held until the application issues a commit transaction. Locks are not automatically released when the next fetch is executed.</p> |
| CUR_OPTCC and CUR_OPTCCVAL- | Rows currently in the buffer are not locked, and other users can update or read them freely.  |

To detect collisions between updates issued by the cursor owner and those issued by other users, cursors save and compare timestamps or column values. Therefore, if you specify either of the optimistic concurrency control options (CUR\_OPTCC or CUR\_OPTCCVAL) your updates can fail because of collisions with other updates. You may want to design the application to refresh the buffer and then retry updates that fail.

The two optimistic concurrency control options differ in the way they detect collisions:

**Table A-4: Detecting concurrency collisions**

| Option       | Method of Detection   |
|--------------|---|
| CUR_OPTCC    | Optimistic concurrency control based on timestamp values. Compares timestamps if available; otherwise, saves and compares the value of all non-text, non-image columns in the table with their previous values. |
| CUR_OPTCCVAL | Optimistic concurrency control based on values. Compares selected values whether or not a timestamp is available.   |

## DB-Library cursor functions

The following list summarizes the DB-Library cursor routines:

| Routine         | Description  |
|-----------------|--|
| dbcursoropen    | Declares and opens the cursor, specifies the size of the fetch buffer and defines the keyset, and sets the concurrency control option. |
| dbcursorinfo    | Returns the number of columns and the number of rows in the open cursor.   |
| dbcursorcolinfo | Returns column information for the specified column number in the open cursor.   |
| dbcursorbind    | Associates program variables with columns.   |
| dbcursorfetch   | Scrolls the fetch buffer.  |
| dbcursor        | Updates, deletes, inserts, and refreshes the rows in the fetch buffer.   |
| dbcursordclose  | Closes the cursor.   |

For details about an individual routine, see its reference page.

## Holding locks

To retain the flexibility of the Adaptive Server Enterprise transaction model, cursors do not automatically issue begin transaction or commit transaction. The duration of locks acquired during cursor operations is entirely under the control of the application. In other words, an application that uses CUR\_LOCKCC on either the dbcursoropen or dbcursor routine must also issue begin transaction for the locking to have any effect.

To hold the lock on the currently buffered rows when CUR\_LOCKCC is used on dbcursorgen, the application must issue commit transaction and begin transaction before each dbcursfetch that scrolls the local buffer (except for the very first dbcursfetch, which should be preceded only by begin transaction).

To use the short-duration locking feature, issue begin transaction before locking the row to be updated with the CUR\_LOCKCC option of dbcurs. If each update is independent, issue commit transaction after each update. If multiple updates to the same screen of data depend on each other, issue commit transaction when the screen is scrolled.

For repeatable-read consistency, specify holdlock in the select statement in dbcursorgen, and issue begin transaction before the first dbcursfetch. Locks are obtained as the data is fetched and are retained until the application issues commit transaction or rollback transaction.

Although you can close and reopen a repeatable-read cursor, you can get the same effect with FETCH\_FIRST.

Other combinations are possible as well. The important thing to remember is that locks are not held unless begin transaction is in effect. Locks acquired while begin transaction is in effect are held until a commit transaction or rollback transaction is issued.

## Stored procedures used by DB-Library cursors

DB-Library's cursor routines call the Adaptive Server Enterprise's catalog stored procedures to find out table formats and identify key columns.

See the *Adaptive Server Enterprise Reference Manual*.



# DB-Library Error Messages

## 20001

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBESYNC   |
| Message text           | Read attempted while out of synchronization with the server. |
| Possible Cause         |  |
| Action/solution        | Contact Sybase Technical Support                             |
| Additional information | Obsolete   |
| Versions               | None   |

## 20002

|                        |                                   |
|------------------------|-----------------------------------|
| Symbolic constant      | SYBEFCON                          |
| Message text           | Server connection failed.         |
| Possible Cause         | Internal I/O error                |
| Action/solution        | Contact Sybase Technical Support. |
| Additional information |                                   |
| Versions               | Earlier than 15.7 ESD #3.         |

## 20003

|                   |                                  |
|-------------------|----------------------------------|
| Symbolic constant | SYBETIME                         |
| Message text      | Server connection timed out.     |
| Possible Cause    | Network I/O operation timed out. |

|                        |   |
|------------------------|---|
| Action/solution        | Contact Sybase Technical Support.   |
| Additional information | An application can increase the amount of time to wait for a server response using dbsettime(). |
| Versions               | All   |

## 20004

|                        |                                   |
|------------------------|-----------------------------------|
| Symbolic constant      | SYBEREAD                          |
| Message text           | Read from the server failed.      |
| Possible cause         | Internal I/O error                |
| Action/solution        | Contact Sybase Technical Support. |
| Additional information |                                   |
| Versions               | All                               |

## 20005

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEUFL   |
| Message text           | DB-LIBRARY internal error - send buffer length corrupted. |
| Possible Cause         | Internal memory error                                     |
| Action/solution        | Contact Sybase Technical Support.                         |
| Additional information |   |
| Versions               | All   |

## 20006

|                   |                             |
|-------------------|-----------------------------|
| Symbolic constant | SYBEWRIT                    |
| Message text      | Write to the server failed. |



|                        |   |
|------------------------|---|
| Possible Cause         | Connection is no longer usable. Server may have stopped responding. |
| Action/solution        | Close and reestablish connection.                                   |
| Additional information |   |
| Versions               | All   |

## 20008

|                        |                                   |
|------------------------|-----------------------------------|
| Symbolic constant      | SYBESOCK                          |
| Message text           | Unable to open socket.            |
| Possible Cause         | Internal I/O error                |
| Action/solution        | Contact Sybase Technical Support. |
| Additional information |                                   |
| Versions               | Earlier than 15.7 ESD #3.         |

## 20009

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECONN   |
| Message text           | Unable to connect socket -- server is unavailable or does not exist. |
| Possible Cause         | Internal I/O error   |
| Action/solution        | Contact Sybase Technical Support.                                    |
| Additional information |  |
| Versions               | All  |

## 20010

|                   |         |
|-------------------|---------|
| Symbolic constant | SYBEMEM |
|-------------------|---------|

|                        |   |
|------------------------|---|
| Message text           | Unable to allocate sufficient memory.         |
| Possible Cause         | Cannot get heap memory.                       |
| Action/solution        | Look at system configuration for heap memory. |
| Additional information |   |
| Versions               | All   |

## 20011

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEDBPS   |
| Message text           | Maximum number of DBPROCESSes already allocated.                 |
| Possible Cause         | The configured maximum number of DBPROCESSes have been exceeded. |
| Action/solution        | Use dbsetmaxprocs() to increase the limit.                       |
| Additional information | The default value is 25.   |
| Versions               | All  |

## 20012

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEINTF  |
| Message text           | Server name not found in interfaces file.                     |
| Possible Cause         | DSQUERY is set incorrectly.                                   |
| Action/solution        | Make sure the server name is included in the interfaces file. |
| Additional information |   |
| Versions               | All   |

## 20013

|                   |          |
|-------------------|----------|
| Symbolic constant | SYBEUHST |
|-------------------|----------|

|                        |  |
|------------------------|--|
| Message text           | Unknown host machine name.                   |
| Possible Cause         | Unknown machine name in the interfaces file. |
| Action/solution        | Correct interfaces file.                     |
| Additional information |  |
| Versions               | All  |

## 20014

|                        |                               |
|------------------------|-------------------------------|
| Symbolic constant      | SYBEPWD                       |
| Message text           | Login incorrect.              |
| Possible Cause         | Incorrect username/ password. |
| Action/solution        | Correct username/ password.   |
| Additional information |                               |
| Versions               | All                           |

## 20015

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEOPIN  |
| Message text           | Could not open interfaces file.                     |
| Possible Cause         | Cannot open interfaces file.                        |
| Action/solution        | Check existence/permissions of the interfaces file. |
| Additional information |   |
| Versions               | All   |

## 20016

|                   |          |
|-------------------|----------|
| Symbolic constant | SYBEINLN |
|-------------------|----------|

|                        |   |
|------------------------|---|
| Message text           | <code>interfaces file: unexpected end-of-line.</code> |
| Possible Cause         | Unexpected end-of-line (EOL) in the interfaces file.  |
| Action/solution        | Ensure that the format of interfaces file is correct. |
| Additional information |   |
| Versions               | All   |

## 20017

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBESEOF                                     |
| Message text           | <code>Unexpected EOF from the server.</code> |
| Possible Cause         | Server has been shut down.                   |
| Action/solution        | Verify server status.                        |
| Additional information |  |
| Versions               | Earlier than 15.7 ESD #3.                    |

## 20018

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBESMSG   |
| Message text           | <code>General server error: Check messages from the server.</code> |
| Possible Cause         | An error occurred on the server.                                   |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20019

|                   |          |
|-------------------|----------|
| Symbolic constant | SYBERPND |
|-------------------|----------|

|                        |  |
|------------------------|--|
| Message Text           | Attempt to initiate a new server operation with results pending.             |
| Possible Cause         | dbsqlexec() has been called before all previous results have been processed. |
| Action/solution        | Process all results with dbresults() before issuing new queries.             |
| Additional information |  |
| Versions               | All  |

## 20020

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEDBPS   |
| Message text           | Bad token from server: Data-stream processing out of sync. |
| Possible Cause         | Internal client/server communication problem.              |
| Action/solution        | Contact Sybase Technical Support.                          |
| Additional information |  |
| Versions               | All  |

## 20021

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEITIM  |
| Message text           | Illegal timeout value specified.                                    |
| Possible Cause         | The value passed to dbsettime() or dbsetlogintime() is invalid.     |
| Action/solution        | Make sure the <i>seconds</i> parameter is a positive integer value. |
| Additional information |   |
| Versions               | All   |

## 20022

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEOOB  |
| Message text           | Error in sending out-of-band data to the server.                         |
| Possible Cause         | Out-of-band data sent by the network failed.                             |
| Action/solution        | Close and then reopen DBPROCESS.   |
| Additional information | Out-of-band data might have been sent as a result of calling dbcancel(). |
| Versions               | All  |

## 20023

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBTYP   |
| Message text           | Unknown bind type passed to DB-LIBRARY function.                                   |
| Possible Cause         | dbbind() (or other bind function) has passed an unknown value for <i>vartype</i> . |
| Action/solution        | Check function documentation for valid bind types and correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20024

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBNCR  |
| Message text           | Attempt to bind user variable to a non-existent compute row.      |
| Possible Cause         | The result set does not contain a compute row.                    |
| Action/solution        | Make sure the query returns compute rows if it calls dbaltbind(). |
| Additional information |   |
| Versions               | All   |

## 20025

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEIICL   |
| Message text           | Illegal integer column length returned by the server.                |
| Possible Cause         | An illegal integer length (other than 1, 2, or 4) was received.      |
| Action/solution        | Unlikely to occur; if from an Open Server, check Open Server coding. |
| Additional information |  |
| Versions               | All  |

## 20026

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECNOR   |
| Message text           | Column number out of range.  |
| Possible Cause         | The column number specified to dbdata() is not part of the result set. |
| Action/solution        | Correct code to reference an available column.                         |
| Additional information |  |
| Versions               | All  |

## 20027

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBENPRM   |
| Message text           | NULL parameter not allowed for this dboption.      |
| Possible Cause         | A NULL parameter has been specified to dbsetopt(). |
| Action/solution        | Correct code to specify valid parameter.           |
| Additional information |  |
| Versions               | All  |

## 20028

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEUVDT   |
| Message text           | Unknown variable-length datatype encountered.                                      |
| Possible Cause         | An unknown variable-length datatype has been received from the server.             |
| Action/solution        | Unlikely to happen; if the server is an Open Server, verify coding of Open Server. |
| Additional information |  |
| Versions               | All  |

## 20029

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEUFDT   |
| Message text           | Unknown fixed-length datatype encountered.   |
| Possible Cause         | An unknown fixed-length datatype has been received from the server.                |
| Action/solution        | Unlikely to happen; if server is an open server, verify coding of the open server. |
| Additional information |  |
| Versions               | All  |

## 20030

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEWAID  |
| Message text           | DB-LIBRARY internal error: ALTFMT following ALTNAME has wrong id. |
| Possible Cause         | Server has sent the wrong ID for a compute row.                   |
| Action/solution        | Contact Sybase Technical Support.                                 |
| Additional information |   |
| Versions               | All   |



## 20031

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECDNS  |
| Message text           | Datastream indicates that a compute column is derived from a non-existent select-list member. |
| Possible Cause         | Compute column is derived from a nonexistent select-list member.                              |
| Action/solution        | Contact Sybase Technical Support.   |
| Additional information |   |
| Versions               | All   |

## 20033

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEABMT  |
| Message text           | User attempted a dbbind() with mismatched column and variable types.                                  |
| Possible Cause         | dbbind() (or other bind function) has been called with incompatible column and <i>vartype</i> values. |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20034

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEABNP                                 |
| Message text           | Attempt to bind using NULL pointers.     |
| Possible Cause         | Bad <i>varaddr</i> argument to dbbind(). |
| Action/solution        | Correct application coding.              |
| Additional information |  |
| Versions               | All                                      |

## 20035

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEAAMT   |
| Message text           | User attempted a dbbind() with mismatched column and variable types. |
| Possible Cause         | Program datatype does not match column datatype.                     |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20036

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBENXID   |
| Message text           | The server did not grant us a distributed-transaction ID.    |
| Possible Cause         | A problem arose during initialization of the commit service. |
| Action/solution        | Verify that the interfaces file has correct entries.         |
| Additional information |  |
| Versions               | All  |

## 20037

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBERXID   |
| Message text           | The server did not recognize our distributed-transaction ID. |
| Possible Cause         | Programming error.   |
| Action/solution        | Validate <i>commid</i> parameter specified to stat_xact().   |
| Additional information |  |
| Versions               | All  |

## 20038

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEICN   |
| Message text           | Invalid computeid or compute column number.                                     |
| Possible Cause         | Invalid <i>computeid</i> or compute column number passed to a dbalt*() routine. |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20039

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBENMOB                                  |
| Message text           | No such member of 'order by' clause.      |
| Possible Cause         | Invalid column ID passed to dbordercol(). |
| Action/solution        | Correct application coding.               |
| Additional information |   |
| Versions               | All                                       |

## 20040

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEAPUT  |
| Message text           | Attempt to print unknown token.                     |
| Possible Cause         | Unknown <i>type</i> parameter passed to dbprtype(). |
| Action/solution        | Correct application coding.                         |
| Additional information |   |
| Versions               | All   |

## 20041

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEASNL   |
| Message text           | Attempt to set fields in a null loginrec.                          |
| Possible Cause         | <i>LOGINREC</i> pointer is NULL.                                   |
| Action/solution        | Call dblogin() to allocate login record.                           |
| Additional information | dblogin() may return a NULL pointer if memory cannot be allocated. |
| Versions               | All  |

## 20042

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBENTLL   |
| Message text           | Name too long for loginrec field.  |
| Possible Cause         | <i>name</i> parameter to DBSETL* routine is longer than <i>DBMAXNAME</i> . |
| Action/solution        | Use a shorter name.  |
| Additional information |  |
| Versions               | All  |

## 20043

|                        |  |
|------------------------|--|
| loginrec==>loginrec    |  |
| Symbolic constant      | SYBEASUL   |
| Message text           | Attempt to set unknown loginrec field.   |
| Possible Cause         | Internal function dbsetlname attempted to set a nonexistent <i>loingrec</i> field. |
| Action/solution        | Contact Sybase Technical Support.  |
| Additional information |  |
| Versions               | All  |

## 20044

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBERDNR  |
| Message text           | Attempt to retrieve data from a non-existent row.         |
| Possible Cause         | dbdata() has been called when there is no data available. |
| Action/solution        | Correct application coding                                |
| Additional information |   |
| Versions               | All   |

## 20045

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBENSI   |
| Message text           | Negative starting index passed to dbstrcpy().           |
| Possible Cause         | <i>start</i> parameter to dbstrcpy() is less than zero. |
| Action/solution        | Correct value.  |
| Additional information |   |
| Versions               | All   |

## 20046

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEABNV  |
| Message text           | Attempt to bind to a NULL program variable.           |
| Possible Cause         | <i>destvar</i> parameter to dbbind is NULL.           |
| Action/solution        | Provide a pointer to memory for the program variable. |
| Additional information |   |
| Versions               | All   |

## 20047

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEDDNE  |
| Message text           | DBPROCESS is dead or not enabled.                       |
| Possible Cause         | An error occurred on the DBPROCESS, making it unusable. |
| Action/solution        | Contact Sybase Technical Support.                       |
| Additional information |   |
| Versions               | All   |

## 20048

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECUFL   |
| Message text           | Data-conversion resulted in underflow.   |
| Possible Cause         | dbconvert() resulted in underflow.   |
| Action/solution        | Correct application coding.  |
| Additional information | The conversion resulted in a loss of precision. If this is unacceptable, rework the application so that the destination variable can fully represent the source value. |
| Versions               | All  |

## 20049

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECOFL   |
| Message text           | Data-conversion resulted in overflow.  |
| Possible Cause         | dbconvert() resulted in overflow.  |
| Action/solution        | Correct application coding.  |
| Additional information | The destination buffer is not large enough to accommodate the converted value. |
| Versions               | All  |

## 20050

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECSYN  |
| Message text           | Attempt to convert data stopped by syntax error in source field.  |
| Possible Cause         | There has been a conversion error.  |
| Action/solution        | Verify that the arguments to dbconvert() are correct. Consult the function documentation to ensure that the source type can be converted to the destination type. |
| Additional information |   |
| Versions               | All   |

## 20051

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECLPR                                       |
| Message text           | Data-conversion resulted in loss of precision. |
| Possible Cause         | dbconvert() call resulted in precision loss.   |
| Action/solution        | Correct application coding.                    |
| Additional information |  |
| Versions               | All  |

## 20052

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECNOV  |
| Message text           | Attempt to set variable to NULL resulted in overflow. |
| Possible Cause         | dbconvert() <i>src</i> parameter should not be NULL.  |
| Action/solution        | Correct application coding.                           |
| Additional information |   |
| Versions               | All   |

## 20053

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBERDCN   |
| Message text           | <code>Requested data-conversion does not exist.</code>                                 |
| Possible Cause         | <code>dbconvert()</code> is unable to convert from <i>srctype</i> to <i>desttype</i> . |
| Action/solution        | Check the datatype conversion table for valid type pairs.                              |
| Additional information |  |
| Versions               | All  |

## 20054

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBESFOV  |
| Message text           | <code>dbsafestr() overflowed its destination buffer.</code> |
| Possible Cause         | Destination buffer is not large enough.                     |
| Action/solution        | Correct application coding.                                 |
| Additional information |   |
| Versions               | All   |

## 20055

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEUNT   |
| Message text           | <code>Unknown network type found in interfaces file.</code> |
| Possible Cause         | Interfaces file has an erroneous server entry.              |
| Action/solution        | Correct interfaces file.                                    |
| Additional information |   |
| Versions               | Earlier than 15.7 ESD #3.                                   |



## 20056

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECLOS   |
| Message text           | Error in closing network connection.             |
| Possible Cause         | There was an error closing the network endpoint. |
| Action/solution        | Contact Sybase Technical Support.                |
| Additional information |  |
| Versions               | All  |

## 20060

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECSYN  |
| Message text           | Unknown datatype encountered.                                     |
| Possible Cause         | A DB-Library routine has been called to pass an unknown datatype. |
| Action/solution        | Correct application coding.                                       |
| Additional information |   |
| Versions               | All   |

## 20061

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBETSIT  |
| Message text           | Attempt to call dbtspu() with an invalid timestamp. |
| Possible Cause         | Adaptive Server column has no timestamp.            |
| Action/solution        | Correct application coding.                         |
| Additional information |   |
| Versions               | All   |

## 20062

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECSYN  |
| Message text           | Attempt to update the timestamp of a table which has no timestamp column. |
| Possible Cause         | dbtsput() has been called on a row that cannot be browsed.                |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20063

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBDIO   |
| Message text           | Bad bulk-copy direction. Must be either IN or OUT.         |
| Possible Cause         | bcp_init() called with invalid <i>direction</i> parameter. |
| Action/solution        | Correct application coding.                                |
| Additional information |  |
| Versions               | All  |

## 20064

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECBNT  |
| Message text           | Attempt to use Bulk Copy with a non-existent server table.  |
| Possible Cause         | <i>table</i> parameter to bcp_init() is invalid for server. |
| Action/solution        | Correct application coding.                                 |
| Additional information |   |
| Versions               | All   |

## 20065

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEIFNB  |
| Message text           | Illegal field number passed to bcp_control().         |
| Possible Cause         | <i>field</i> parameter to bcp_control() is incorrect. |
| Action/solution        | Correct application coding.                           |
| Additional information |   |
| Versions               | All   |

## 20066

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBETTS   |
| Message text           | The table which bulk-copy is attempting to copy to a host-file is shorter than the number of rows which bulk-copy was instructed to skip. |
| Possible Cause         | dbcontrol() was called with a BCPFIRST value greater than the number of rows in the table.  |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20067

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEKBCO  |
| Message text           | 1000 rows successfully bulk-copied to host-file.  |
| Possible Cause         | 1000 rows copied.   |
| Action/solution        | No action required.   |
| Additional information | This is an informational message informing the user of the progress in copying rows to the host file. |
| Versions               | All   |

## 20068

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBBCI  |
| Message text           | Batch successfully bulk-copied to the server.                                   |
| Possible Cause         | A batch has been successfully copied.   |
| Action/solution        | No action required.   |
| Additional information | This is an informational message informing the user of the progress in copying. |
| Versions               | All   |

## 20069

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEKBCI  |
| Message text           | Bcp:1000 rows sent to the server.   |
| Possible Cause         | Rows have been sent to the server.  |
| Action/solution        | No action required.   |
| Additional information | This is an informational message informing the user of the progress in copying. |
| Versions               | All   |

## 20070

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBCRE                               |
| Message text           | I/O error while reading bcp data-file. |
| Possible Cause         | System I/O routing failed.             |
| Action/solution        | Contact Sybase Technical Support.      |
| Additional information |  |
| Versions               | All                                    |

## 20071

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBETPTN  |
| Message text           | Syntax error:only two periods are permitted in table names. |
| Possible Cause         | Table reference syntax is incorrect.                        |
| Action/solution        | Correct query.  |
| Additional information |   |
| Versions               | All   |

## 20072

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECWE                                |
| Message text           | I/O error while writing bcp data-file. |
| Possible Cause         | System I/O routing failed.             |
| Action/solution        | Contact Sybase Technical Support.      |
| Additional information |  |
| Versions               | All                                    |

## 20073

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECNN  |
| Message text           | Attempt to bulk-copy a NULL value into server column <colname>, which does not accept NULL values. |
| Possible Cause         | Incorrect insert/update query.   |
| Action/solution        | Correct query.   |
| Additional information |  |
| Versions               | All  |

## 20074

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBCOR  |
| Message text           | Attempt to bulk-copy an oversized row to the server.  |
| Possible Cause         | Data to be copied is too large for the server column. |
| Action/solution        | Reconcile <i>bcp</i> file/server schema.              |
| Additional information |   |
| Versions               | All   |

## 20075

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEB CIS   |
| Message text           | Attempt to bulk-copy an illegally-sized column value to the server. |
| Possible Cause         | Data to be copied is too large for the server column.               |
| Action/solution        | Reconcile <i>bcp</i> file/server schema.                            |
| Additional information |   |
| Versions               | All   |

## 20076

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEB CPI  |
| Message text           | <code>bcp_init()</code> must be called before any other <i>bcp</i> routines. |
| Possible Cause         | <code>bcp_init()</code> not called.  |
| Action/solution        | Call <code>bcp_init()</code> .   |
| Additional information |  |
| Versions               | All  |

## 20077

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBCOR   |
| Message text           | <code>bcp_bind()</code> , <code>bcp_collen()</code> and <code>bcp_colptr()</code> may be used only after <code>bcp_init()</code> has been called with the copy direction set to <code>DB_IN</code> . |
| Possible Cause         | Above functions called before <code>bcp_init()</code> .  |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20078

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBCPB  |
| Message text           | <code>bcp_bind()</code> may NOT be used after <code>bcp_init()</code> has been passed a non-NULL input file name. |
| Possible Cause         | Attempting to copy from a program variable when an input file has been specified.                                 |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20079

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEVDPT  |
| Message text           | For bulk copy, all variable-length data must have either a length-prefix or a terminator specified. |
| Possible Cause         | Missing <i>length-prefix</i> or <i>terminator</i> .   |
| Action/solution        | Correct parameters to <code>bcp_colfmt()</code> .   |
| Additional information |   |

Versions All

## 20080

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBIVI  |
| Message text           | <code>bcp_columns()</code> and <code>bcp_colfmt()</code> may be used only after <code>bcp_init()</code> has been passed a valid input file. |
| Possible Cause         | <code>bcp_init()</code> has been called with an invalid <i>hfile</i> parameter.   |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20081

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEECBC  |
| Message text           | <code>bcp_columns()</code> must be called before <code>bcp_colfmt()</code> .          |
| Possible Cause         | <code>bcp_colfmt()</code> has been called before calling <code>bcp_columns()</code> . |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20082

|                   |  |
|-------------------|--|
| Symbolic constant | SYBEEBCFO  |
| Message text      | Bcp host-files must contain at least one column.   |
| Possible Cause    | <code>bcp_columns()</code> has been called with an invalid <i>host_colcount</i> parameter. |
| Action/solution   | Correct application coding.  |



Additional information

Versions All

## 20083

Symbolic constant SYBEBCVH

Message text `bcp_exec()` may be called only after `bcp_init()` has been passed a valid host file.

Possible Cause `bcp_init()` has been called with an invalid *hfile* parameter.

Action/solution Correct application coding.

Additional information

Versions All

## 20084

Symbolic constant SYBEBCUO

Message text `Bcp: Unable to open host data-file.`

Possible Cause `bcp_init()` has been called with an invalid *hfile* parameter.

Action/solution Correct application coding.

Additional information

Versions All

## 20085

Symbolic constant SYBEBCUC

Message text `Bcp: Unable to close host data-file.`

Possible Cause System close routine failed.

Action/solution                      Contact Sybase Technical Support.

Additional information

Versions                              All

## 20086

Symbolic constant                      SYBEBUOE

Message text                              Bcp: Unable to open error-file.

Possible Cause                              bcp\_init() has been called with an invalid *errfile* parameter.

Action/solution                              Correct application coding.

Additional information

Versions                              All

## 20087

open==>close

Symbolic constant                      SYBEBUCE

Message text                              Bcp: Unable to open error-file.

Possible Cause                              System close routine failed.

Action/solution                              Contact Sybase Technical Support.

Additional information

Versions                              All

## 20088

Symbolic constant                      SYBEBWEF

Message text                              I/O error while writing bcp error-file.

|                        |                                   |
|------------------------|-----------------------------------|
| Possible Cause         | System I/O routine failed.        |
| Action/solution        | Contact Sybase Technical Support. |
| Additional information |                                   |
| Versions               | All                               |

## 20091

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEASEC   |
| Message text           | Attempt to send an empty command buffer to the server.               |
| Possible Cause         | dbcmd() has not been called or has been called with an empty string. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20092

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBETMTD   |
| Message text           | Attempt to send too much TEXT data via the dbmoretext() call.  |
| Possible Cause         | dbwritetext() specifies the total text length to be set. This error indicates that the cumulative total sent by dbmoretext() calls exceed the value specified in the dbwritetext() call. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20093

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBENTTN   |
| Message text           | Attempt to use dbtxtsput() to put a new text-timestamp into a non-existent data row. |
| Possible Cause         | dbtxtsput() called incorrectly.  |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20094

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEDNTI   |
| Message text           | Attempt to use dbtxtsput() to put a new text-timestamp into a column whose datatype is neither SYBTEXT nor SYBIMAGE. |
| Possible Cause         | dbtxtsput() called incorrectly.  |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20095

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBTMT   |
| Message text           | Attempt to send too much TEXT data via the bcp_moretext() call.            |
| Possible Cause         | The size of the text sent by bcp_moretext() calls exceeds the column size. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20096

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEORPF  |
| Message text           | Attempt to set remote password would overflow the login-record's remote-password field.   |
| Possible Cause         | Remote password is too long.  |
| Action/solution        | Correct application coding.   |
| Additional information | The remote password buffer is 255 bytes long. Each password's entry in the buffer consists of the password itself, the associated server name, and 2 extra bytes. |
| Versions               | All   |

## 20097

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEUVBF   |
| Message text           | Attempt to read an unknown version of BCP format-file. |
| Possible Cause         | Format file has been corrupted.                        |
| Action/solution        | Contact Sybase Technical Support.                      |
| Additional information |  |
| Versions               | All  |

## 20098

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBUOF   |
| Message text           | Bcp: Unable to open format-file.                             |
| Possible Cause         | Incorrect <i>filename</i> parameter passed to bcp_readfmt(). |
| Action/solution        | Correct application coding.                                  |
| Additional information |  |
| Versions               | All  |

## 20099

|                        |                                   |
|------------------------|-----------------------------------|
| Symbolic constant      | SYBEBUCF                          |
| Message text           | Bcp: Unable to close format-file. |
| Possible Cause         | System close routine failed.      |
| Action/solution        | Contact Sybase Technical Support. |
| Additional information |                                   |
| Versions               | All                               |

## 20100

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBRFF                                 |
| Message text           | I/O error while reading bcp format-file. |
| Possible Cause         | System I/O routine failed.               |
| Action/solution        | Contact Sybase Technical Support.        |
| Additional information |  |
| Versions               | All                                      |

## 20101

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBWFF                                 |
| Message text           | I/O error while writing bcp format-file. |
| Possible Cause         | System I/O routine failed.               |
| Action/solution        | Contact Sybase Technical Support.        |
| Additional information |  |
| Versions               | All                                      |

## 20102

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBUDF   |
| Message text           | Bcp: Unrecognized datatype found in format-file. |
| Possible Cause         | Corrupted input file.                            |
| Action/solution        | Contact Sybase Technical Support.                |
| Additional information |  |
| Versions               | All  |

## 20103

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBIHC   |
| Message text           | Incorrect host-column number found in bcp format-file. |
| Possible Cause         | Corrupted input file.                                  |
| Action/solution        | Contact Sybase Technical Support.                      |
| Additional information |  |
| Versions               | All  |

## 20104

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBEOF                                     |
| Message text           | Unexpected EOF encountered in BCP data-file. |
| Possible Cause         | Corrupted input file.                        |
| Action/solution        | Contact Sybase Technical Support.            |
| Additional information |  |
| Versions               | All  |

## 20105

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBCNL                                       |
| Message text           | Negative length-prefix found in BCP data-file. |
| Possible Cause         | Corrupted input file.                          |
| Action/solution        | Contact Sybase Technical Support.              |
| Additional information |  |
| Versions               | All  |

## 20106

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBCSI  |
| Message text           | Host-file columns may be skipped only when copying Into the server. |
| Possible Cause         | Programming error.  |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20107

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEB CIT   |
| Message text           | It's illegal to use BCP terminators with program variables other than SYBCHAR, SYBBINARY, SYBTEXT, or SYBIMAGE. |
| Possible Cause         | bcp_bind() called incorrectly.  |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |



## 20108

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBCSA  |
| Message text           | The BCP hostfile <filename> contains only <n> rows.<br>Skipping all of these rows is not allowed. |
| Possible Cause         | bcf_control() has set BCPFIRST to a value greater than the number of rows in the input file.      |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20109

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBENULL  |
| Message text           | NULL DBPROCESS pointer passed to DB-Library.          |
| Possible Cause         | A DB-Library routine passed a NULL DBPROCESS pointer. |
| Action/solution        | Correct application coding.                           |
| Additional information |   |
| Versions               | All   |

## 20110

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEUNAM  |
| Message text           | Unable to get current username from operating system. |
| Possible Cause         | getpwuid() system call failed.                        |
| Action/solution        | Contact your system administrator.                    |
| Additional information |   |
| Versions               | All   |

## 20111

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBCRO  |
| Message text           | The BCP hostfile <filename> contains only <n> rows. It was impossible to read the requested <m> rows. |
| Possible Cause         | bcf_control() has set <i>BCPLAST</i> to a value greater than the number of rows in the input file.    |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20112

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEMPLL   |
| Message text           | Attempt to set maximum number of DBPROCESSes lower than 1. |
| Possible Cause         | dbsetmaxprocs() called with a value less than 1.           |
| Action/solution        | Correct application coding.                                |
| Additional information |  |
| Versions               | All  |

## 20113

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBERPIL  |
| Message text           | It is illegal to pass -1 to dbrpcparam() for the datalen of parameters which are of type SYBCHAR, SYBVARCHAR, SYBBINARY, or SYBVARBINARY. |
| Possible Cause         | Incorrect data length supplied for variable length datatype.  |
| Action/solution        | Correct application coding.   |
| Additional information |   |

Versions All

## 20114

Symbolic constant SYBERPUL

Message text When passing a SYBINTN, SYBDATETIMN, SYBMONEYN, or SYBFLTNN parameter via `rpcparam()`, it's necessary to specify the parameter's maximum or actual length, so that DB-Library can recognize it as a SYBINT1, SYBINT2, SYBINT4, SYBMONEY, SYBMONEY4, etc.

Possible Cause `dbrpcparam()` called with incorrect *datlen* value.

Action/solution Correct application coding.

Additional information

Versions All

## 20115

Symbolic constant SYBEUNOP

Message text Unknown option passed to `dbsetopt()`.

Possible Cause `dbsetopt()` called incorrectly.

Action/solution Correct application coding.

Additional information

Versions All

## 20116

Symbolic constant SYBECRNC

Message text The current row is not a result of compute clause <computeid>, so it is illegal to attempt to extract that

|                        |  |
|------------------------|--|
|                        | <code>data from this row.</code>   |
| Possible Cause         | <code>dbadata()</code> has been called on a row that is not the result of an alter row clause. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20117

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBERTCC  |
| Message text           | <code>dbreadtext()</code> may not be used to receive the results of a query which contains a <code>COMPUTE</code> clause. |
| Possible Cause         | <code>dbreadtext()</code> has been called to retrieve text from the result of an alter row clause.                        |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20118

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBERTSC  |
| Message text           | <code>dbreadtext()</code> may only be used to receive the results of a query which contains a single result column. |
| Possible Cause         | <code>dbreadtext()</code> has been called on a result set containing more than one column.                          |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20119

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEUCRR   |
| Message text           | Internal software error: Unknown connection result reported by dbpasswd(). |
| Possible Cause         | There is a problem connecting to the server.                               |
| Action/solution        | Contact Sybase Technical Support.  |
| Additional information |  |
| Versions               | All  |

## 20120

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBERPNA   |
| Message text           | The RPC facility is available only when using a server whose version number is 4.0 or greater. |
| Possible Cause         | This version of Adaptive Server does not support RPCs.   |
| Action/solution        | Upgrade Adaptive Server.   |
| Additional information |  |
| Versions               | All  |

## 20121

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEOPNA  |
| Message text           | The text/image facility is available only when using a server whose version number is 4.0 or greater. |
| Possible Cause         | This version of Adaptive Server does not support text/image.  |
| Action/solution        | Upgrade Adaptive Server.  |
| Additional information |   |
| Versions               | All   |

## 20122

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEFGTL   |
| Message text           | Bcp: Row number of the first row to be copied cannot be greater than the row number for the last row to be copied. |
| Possible Cause         | bcp_control() has been called with inconsistent <i>BCPFIRST</i> and <i>BCPLAST</i> .                               |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20123

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECWLL                                 |
| Message text           | Attempt to set column width less than 1. |
| Possible Cause         |  |
| Action/solution        | Contact Sybase Technical Support.        |
| Additional information | Obsolete                                 |
| Versions               | None                                     |

## 20124

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEUFDS   |
| Message text           | Unrecognized format encountered in dbstrbuild().   |
| Possible Cause         | A format specifier that does not match any custom-installed via obsolete function dbfmtinstall() specified in the format string. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20125

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEUCPT   |
| Message text           | Unrecognized custom-format parameter-type encountered in dbstrbuild(). |
| Possible Cause         | dbstrbuild() called with invalid parameter type in format string.      |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20126

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBETMCF   |
| Message text           | Attempt to install too many custom formats via (obsolete function) dbfmtinstall(). |
| Possible Cause         | dbfmtinstall() called with more than <i>MAXFMTS</i> (20) formats.                  |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20127

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEAICF                                      |
| Message text           | Error in attempting to install custom format. |
| Possible Cause         | dbfmtinstall() generic failure.               |
| Action/solution        | Examine application code.                     |
| Additional information |   |
| Versions               | All   |

## 20128

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEADST   |
| Message text           | Error in attempting to determine the size of a pair of translation tables. |
| Possible Cause         | dbload_xlate() called with invalid <i>srv_charset</i> .                    |
| Action/solution        | Correct application coding.  |
| Additional information | Make sure specified character set exists in <i>\$\$SYBASE/charsets</i> .   |
| Versions               | All  |

## 20129

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEALTT  |
| Message text           | Error in attempting to load a pair of translation tables. |
| Possible Cause         | Internal error loading translation tables.                |
| Action/solution        | Contact tech support.                                     |
| Additional information |   |
| Versions               | All   |

## 20130

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEAPCT  |
| Message text           | Error in attempting to perform a character-set translation. |
| Possible Cause         | dbxlate() has failed.                                       |
| Action/solution        | Check parameters to dbxlate().                              |
| Additional information |   |
| Versions               | All   |



## 20131

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEXOCI   |
| Message text           | A character-set translation overflowed its destination buffer while using bcp to copy data from a host-file to the server. |
| Possible Cause         | Internal conversion error.   |
| Action/solution        | Contact Sybase Technical Support.  |
| Additional information |  |
| Versions               | All  |

## 20132

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEFSHD   |
| Message text           | Error in attempting to find the Sybase home directory.   |
| Possible Cause         | <code>\$SYBASE</code> environment variable is incorrect. |
| Action/solution        | Correct setting.   |
| Additional information |  |
| Versions               | All  |

## 20133

|                        |  |
|------------------------|--|
| Symbolic constant      |  |
| Message text           | Error in attempting to open a localization file. |
| Possible Cause         | Unable to open a localization file.              |
| Action/solution        | Make sure <code>\$SYBASE</code> is correct.      |
| Additional information |  |
| Versions               | All  |

## 20134

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEARDI   |
| Message text           | Error in attempting to read datetime information from a localization file. |
| Possible Cause         | Corrupt <i>locales</i> file.   |
| Action/solution        | Contact Sybase Technical Support.  |
| Additional information |  |
| Versions               | All  |

## 20135

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEURCI   |
| Message text           | Unable to read copyright information\from the dblib localization file. |
| Possible Cause         | Corrupt <i>locales</i> file.   |
| Action/solution        | Contact Sybase Technical Support.                                      |
| Additional information |  |
| Versions               | All  |

## 20136

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEARDL   |
| Message text           | Error in attempting to read the dblib.loc localization file. |
| Possible Cause         | Corrupt <i>locales</i> file.                                 |
| Action/solution        | Contact Sybase Technical Support.                            |
| Additional information |  |
| Versions               | All  |

## 20137

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEURMI  |
| Message text           | Unable to read money-format information from the dblib localization file. |
| Possible Cause         | Corrupt <i>locales</i> file.  |
| Action/solution        | Contact Sybase Technical Support.   |
| Additional information |   |
| Versions               | All   |

## 20138

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEUREM  |
| Message text           | Unable to read error mnemonic from the dblib localization file. |
| Possible Cause         | Corrupt <i>locales</i> file.                                    |
| Action/solution        | Contact Sybase Technical Support.                               |
| Additional information |   |
| Versions               | All   |

## 20139

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEURES  |
| Message text           | Unable to read error string from the dblib localization file. |
| Possible Cause         | Corrupt <i>locales</i> file.                                  |
| Action/solution        | Contact Sybase Technical Support.                             |
| Additional information |   |
| Versions               | All   |

## 20140

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEUREI  |
| Message text           | Unable to read error information from the dblib<br>localization file. |
| Possible Cause         | Corrupt <i>locales</i> file.  |
| Action/solution        | Contact Sybase Technical Support.                                     |
| Additional information |   |
| Versions               | All   |

## 20141

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEOREN  |
| Message text           | Warning: an out-of-range error-number was encountered<br>in dblib.loc. The maximum permissible error-number is<br>defined as DBERRCOUNT in sybdb.h. |
| Possible Cause         | Corrupt <i>locales</i> file.  |
| Action/solution        | Contact Sybase Technical Support.   |
| Additional information |   |
| Versions               | All   |

## 20142

|                        |                                       |
|------------------------|---------------------------------------|
| Symbolic constant      | SYBEISOI                              |
| Message text           | Invalid sort-order information found. |
| Possible Cause         | Corrupt <i>locales</i> file.          |
| Action/solution        | Contact Sybase Technical Support.     |
| Additional information |                                       |
| Versions               | All                                   |

## 20143

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEIDCL  |
| Message text           | Illegal datetime column length returned by the server.<br>Legal datetime lengths are 4 and 8 bytes. |
| Possible Cause         | Internal Adaptive Server error.   |
| Action/solution        | Contact Sybase Technical Support.   |
| Additional information |   |
| Versions               | All   |

## 20144

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEIMCL  |
| Message text           | Illegal money column length returned by the server.<br>Legal money lengths are 4 and 8 bytes. |
| Possible Cause         | Internal Adaptive Server error.   |
| Action/solution        | Contact Sybase Technical Support.   |
| Additional information |   |
| Versions               | All   |

## 20145

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEIFCL  |
| Message text           | Illegal floating-point column length returned by the<br>server. Legal floating-point lengths are 4 and 8 bytes. |
| Possible Cause         | Internal Adaptive Server error.   |
| Action/solution        | Contact Sybase Technical Support.   |
| Additional information |   |
| Versions               | All   |

## 20146

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEUTDS   |
| Message text           | Unrecognized TDS version received from the server. |
| Possible Cause         | Internal error.                                    |
| Action/solution        | Contact Sybase Technical Support.                  |
| Additional information |  |
| Versions               | All  |

## 20147

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBUCF   |
| Message text           | Bcp: Unable to create format-file.                       |
| Possible Cause         | Insufficient access rights.                              |
| Action/solution        | Check access rights.                                     |
| Additional information | This error is raised for both create and close failures. |
| Versions               | All  |

## 20148

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEACNV  |
| Message text           | Attempt to do data-conversion with NULL destination variable. |
| Possible Cause         | Null pointer passed to dbconvert() or dbdatechar().           |
| Action/solution        | Correct application coding.                                   |
| Additional information |   |
| Versions               | All   |

## 20149

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEDPOR  |
| Message text           | Out-of-range datepart constant.                               |
| Possible Cause         | Invalid <i>datepart</i> passed to <code>dbdatechar()</code> . |
| Action/solution        | Correct application coding.                                   |
| Additional information |   |
| Versions               | All   |

## 20150

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBENDC   |
| Message text           | Cannot have negative component in the date in numeric form. |
| Possible Cause         | Invalid <i>value</i> passed to <code>dbdatechar()</code> .  |
| Action/solution        | Correct application coding.                                 |
| Additional information |   |
| Versions               | All   |

## 20151

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEMVOR   |
| Message text           | Month values must be between 1 and 12.                           |
| Possible Cause         | Invalid <i>month</i> value passed to <code>dbdatechar()</code> . |
| Action/solution        | Correct application coding.                                      |
| Additional information |  |
| Versions               | All  |

## 20152

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEDVOR   |
| Message text           | Day values must be between 1 and 7.              |
| Possible Cause         | Invalid <i>day</i> value passed to dbdatechar(). |
| Action/solution        | Correct application coding.                      |
| Additional information |  |
| Versions               | All  |

## 20153

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBENBVP  |
| Message text           | Cannot pass dbsetnull() a NULL bindval pointer.   |
| Possible Cause         | Null pointer passed as <i>bindval</i> .   |
| Action/solution        | Correct application coding.   |
| Additional information | When a null value is encountered, <i>bindval</i> is the value to use instead. NULL is inappropriate here. |
| Versions               | All   |

## 20154

|                        |                                     |
|------------------------|-------------------------------------|
| Symbolic constant      | SYBESPID                            |
| Message text           | Called dbspid() with a NULL dbproc. |
| Possible Cause         | Incorrect coding.                   |
| Action/solution        | Correct application coding.         |
| Additional information |                                     |
| Versions               | All                                 |



## 20155

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBENDTP   |
| Message text           | Called dbdatecrack() with NULL datetime parameter. |
| Possible Cause         | Incorrect coding.                                  |
| Action/solution        | Correct application coding.                        |
| Additional information |  |
| Versions               | All  |

## 20156

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEXTN  |
| Message text           | The xlt_tosrv and xlt_todisp parameters to dbfree_xlate() were NULL. |
| Possible Cause         | Invalid null parameters passed.                                      |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20157

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEXTDN  |
| Message text           | Warning: the xlt_todisp parameter to dbfree_xlate() was NULL. The space associated with the xlt_tosrv parameter has been freed. |
| Possible Cause         | Null <i>xlt_todisp</i> parameter passed to dbfree_xlate().  |
| Action/solution        | Do not free <i>xlt_tosrv</i> .  |
| Additional information |   |
| Versions               | All   |

## 20158

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEXTSN   |
| Message text           | Warning: the <code>xlt_tosrv</code> parameter to <code>dbfree_xlate()</code> was NULL. The space associated with the <code>xlt_todisp</code> parameter has been freed. |
| Possible Cause         | Null <i>xlt_tosrv</i> parameter passed to <code>dbfree_xlate()</code> .  |
| Action/solution        | Do not free <i>xlt_todisp</i> .  |
| Additional information |  |
| Versions               | All  |

## 20159

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBENUM   |
| Message text           | Incorrect number of arguments given to DB-Library.        |
| Possible Cause         | Wrong number of arguments passed to a DB-Library routine. |
| Action/solution        | Correct application coding.                               |
| Additional information |   |
| Versions               | All   |

## 20160

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBETYPE   |
| Message text           | Invalid argument type given to Hyper/DB-Library.       |
| Possible Cause         | <code>dbregparam()</code> called with an invalid type. |
| Action/solution        | Correct application coding.                            |
| Additional information |  |
| Versions               | All  |

## 20161

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEGENOS                                       |
| Message text           | General operating system error.                 |
| Possible Cause         | Creating an internal condition variable failed. |
| Action/solution        | Contact Sybase Technical Support.               |
| Additional information |   |
| Versions               | All   |

## 20162

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEPAGE  |
| Message text           | Wrong resource type or length given for dbpage??() operation. |
| Possible Cause         |   |
| Action/solution        | Contact Sybase Technical Support.                             |
| Additional information | Obsolete.   |
| Versions               | None  |

## 20163

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEOPTNO                                |
| Message text           | Option not allowed.                      |
| Possible Cause         | Internal failure to set security option. |
| Action/solution        | Contact Sybase Technical Support.        |
| Additional information |  |
| Versions               | All                                      |

## 20164

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEETD  |
| Message text           | Failure to send the expected amount of TEXT or IMAGE data via dbmoretext().            |
| Possible Cause         | Application attempted to read results from server before completely sending text data. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20165

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBERTYPE  |
| Message text           | Invalid resource type given to Hyper/DB-Library. |
| Possible Cause         |  |
| Action/solution        | Contact Sybase Technical Support.                |
| Additional information | Obsolete   |
| Versions               | None   |

## 20166

|                        |                                   |
|------------------------|-----------------------------------|
| Symbolic constant      | SYBERFILE                         |
| Message text           | Cannot open resource file.        |
| Possible Cause         |                                   |
| Action/solution        | Contact Sybase Technical Support. |
| Additional information | Obsolete                          |
| Versions               | None                              |

## 20167

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEFMODE   |
| Message text           | Read/Write/Append mode denied on file.                            |
| Possible Cause         | dbstrbuild() called with invalid parameter type in format string. |
| Action/solution        | Correct application coding.                                       |
| Additional information | Obsolete  |
| Versions               | None  |

## 20168

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBESLCT                                  |
| Message text           | Could not select or copy field specified. |
| Possible Cause         |   |
| Action/solution        | Contact Sybase Technical Support.         |
| Additional information | Obsolete                                  |
| Versions               | None                                      |

## 20169

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEZTXT   |
| Message text           | Attempt to send zero length TEXT or IMAGE to dataserver via dbwritetext(). |
| Possible Cause         | Invalid arguments in dbwritetext() call.                                   |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20170

|                        |   |
|------------------------|---|
| Message type           | Error   |
| Symbolic constant      | SYBENTST  |
| Message text           | VMS: The file being opened must be a stream_lf. |
| Possible Cause         |   |
| Action/solution        | Contact Sybase Technical Support.               |
| Additional information | Obsolete  |
| Versions               | None  |

## 20171

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEOSSL   |
| Message text           | Login incorrect: operating system login security level is not within the range of the secure server. |
| Possible Cause         |  |
| Action/solution        | Contact Sybase Technical Support.  |
| Additional information | Obsolete   |
| Versions               | None   |

## 20172

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEESSL  |
| Message text           | Login incorrect: login security level as entered does not agree with your operating system level. |
| Possible Cause         |   |
| Action/solution        | Contact Sybase Technical Support.   |
| Additional information | Obsolete  |
| Versions               | None  |

## 20173

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBENLNL   |
| Message text           | Program not linked with specified network library. |
| Possible Cause         |  |
| Action/solution        | Contact Sybase Technical Support.                  |
| Additional information | Obsolete   |
| Versions               | None   |

## 20174

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBENHAN  |
| Message text           | Called dbrecvpassthru() with a NULL handle parameter. |
| Possible Cause         | dbrecvpassthru() called with invalid parameter.       |
| Action/solution        | Correct application coding.                           |
| Additional information |   |
| Versions               | All   |

## 20175

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBENBUF   |
| Message text           | Called dbsendpassthru() with a NULL buf parameter. |
| Possible Cause         | dbsendpassthru() called with invalid parameter.    |
| Action/solution        | Correct application coding.                        |
| Additional information |  |
| Versions               | All  |

## 20176

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBENULP   |
| Message text           | Called <routine> with a NULL <paramname> parameter.                  |
| Possible Cause         | A DB-Library routine has been called with an invalid NULL parameter. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20177

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBENOTI  |
| Message text           | An event handler must be installed before a notification request can be made. |
| Possible Cause         |   |
| Action/solution        | Contact Sybase Technical Support.   |
| Additional information | Obsolete  |
| Versions               | None  |

## 20178

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEEVOP  |
| Message text           | Called dbregwatch() with a bad options parameter. |
| Possible Cause         | Incorrect options value passed to dbregwatch().   |
| Action/solution        | Correct application coding.                       |
| Additional information |   |
| Versions               | All   |



## 20179

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBENEHA  |
| Message text           | Called dbreghandle() with a NULL handler parameter. |
| Possible Cause         |   |
| Action/solution        | Contact Sybase Technical Support.                   |
| Additional information | Obsolete  |
| Versions               | None  |

## 20180

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBETRAN   |
| Message text           | DBPROCESS is being used for another transaction.                       |
| Possible Cause         | Processing of the previous command on this DBPROCESS is not completed. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20181

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEEVST   |
| Message text           | Must initiate a transaction before calling dbregparam().                               |
| Possible Cause         | dbreginit() or dbregparam() has not been called before the invocation of dbregparam(). |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20182

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEEINI  |
| Message text           | <code>Must call dbreginit() before dbregexec().</code>                |
| Possible Cause         | dbreginit() has not been called before the invocation of dbregexec(). |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20183

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEECRT   |
| Message text           | <code>Must call dbnpdefine() before dbnpcreate().</code> |
| Possible Cause         |  |
| Action/solution        | Contact Sybase Technical Support.                        |
| Additional information | Obsolete   |
| Versions               | None   |

## 20184

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEECAN   |
| Message text           | <code>Attempted to cancel unrequested event notification.</code> |
| Possible Cause         | dbregnowatch() has been called with no prior dbregwatch().       |
| Action/solution        | Correct application coding.                                      |
| Additional information |  |
| Versions               | All  |

## 20185

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEEUNR   |
| Message text           | Unsolicited event notification received.   |
| Possible Cause         | dbreghandle() has been called to uninstall an event handler, but the notification request has not been cancelled by calling dbregnowatch(). When the event is raised, there is no handler installed. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20186

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBERPCS   |
| Message text           | Must call dbrpcinit() before dbrpcparam() or dbrpcsend().                          |
| Possible Cause         | dbrpcinit() has not been called prior to this call to dbrpcparam() or dbrpcsend(). |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20187

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBETPAR  |
| Message text           | No SYBTEXT or SYBIMAGE parameters were defined.   |
| Possible Cause         | dbwritetext() or dbmoretext() called during an RPC with no text/image parameters defined. |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20188

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBETEXS  |
| Message text           | <code>Called dbmoretext() with a bad size parameter.</code>   |
| Possible Cause         | <code>dbmoretext()</code> or <code>dbreadtext()</code> called with a negative <i>bufsize</i> parameter. |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20189

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBETRAC  |
| Message text           | <code>Attempted to turn off a trace flag that was not on.</code>        |
| Possible Cause         | <code>dbtraceoff()</code> called with an invalid <i>flag</i> parameter. |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20190

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBETRAS  |
| Message text           | <code>DB-Library internal error - trace structure not found.</code> |
| Possible Cause         | There is no trace record in the DBPROCESS structure.                |
| Action/solution        | Contact Sybase Technical Support.                                   |
| Additional information |   |
| Versions               | All   |

## 20191

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEPRTF  |
| Message text           | <code>dbtracestring()</code> may only be called from a <code>printfunc()</code> .   |
| Possible Cause         | <code>dbtracestring()</code> called directly instead of from <code>printfunc()</code> set with <code>dbtraceon()</code> . |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20192

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBETRSN   |
| Message text           | <code>Bad numbytes parameter passed to dbtracestring()</code> .              |
| Possible Cause         | The <i>num</i> parameter passed to <code>dbtracestring()</code> is negative. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20193

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBPKS  |
| Message text           | <code>In DBSETLPACKET(), the packet size parameter must be between 256 and 9999.</code> |
| Possible Cause         | Invalid size passed to <code>DBSETLPACKET()</code> .                                    |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20194

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEIPV  |
| Message text           | <code>&lt;value&gt; is an illegal value for the &lt;paramname&gt;<br/>parameter of &lt;routine&gt;.</code> |
| Possible Cause         | Parameter value is outside the domain of the parameter.  |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20195

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEMOV   |
| Message text           | <code>Money arithmetic resulted in overflow in function<br/>&lt;routine&gt;.</code> |
| Possible Cause         | Invalid parameter passed to a dbmny*() function.                                    |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20196

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEDIVZ  |
| Message text           | <code>Attempt to divide by \$0.00 in function &lt;routine&gt;.</code> |
| Possible Cause         | Invalid value passed to dbmydiv() or dbmny4div().                     |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20197

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEASTL                                |
| Message text           | Synchronous I/O attempted at AST level. |
| Possible Cause         |   |
| Action/solution        | Contact Sybase Technical Support.       |
| Additional information | Obsolete VMS-specific error message     |
| Versions               | None                                    |

## 20198

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBESEFA   |
| Message text           | DB_SETEVENT_VMS cannot be called if connections are present. |
| Possible Cause         |  |
| Action/solution        | Contact Sybase Technical Support.                            |
| Additional information | Obsolete VMS-specific error message                          |
| Versions               | None   |

## 20199

|                        |                                      |
|------------------------|--------------------------------------|
| Symbolic constant      | SYBEPOLL                             |
| Message text           | There is already an active dbpoll(). |
| Possible Cause         |                                      |
| Action/solution        | Contact Sybase Technical Support.    |
| Additional information | Obsolete VMS-specific error message  |
| Versions               | None                                 |

## 20200

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBENOEV  |
| Message text           | DBPOLL cannot be called when registered procedure notifications have been disabled. |
| Possible Cause         |   |
| Action/solution        | Contact Sybase Technical Support.   |
| Additional information | Obsolete VMS-specific error message   |
| Versions               | None  |

## 20201

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBADPK   |
| Message text           | Packet size of <requested size> not supported -- size of <other size> used instead. |
| Possible Cause         | DB-Library is cannot accommodate requested packet size.                             |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20202

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBESECURE  |
| Message text           | Secure server function not supported in this version. |
| Possible Cause         | Obsolete routine DBSETLHIER() has been called.        |
| Action/solution        | Correct application coding.                           |
| Additional information |   |
| Versions               | All   |



## 20203

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECAP   |
| Message text           | DB-Library capabilities not accepted by the server. |
| Possible Cause         | Invalid TDS received from server.                   |
| Action/solution        | Contact Sybase Technical Support.                   |
| Additional information |   |
| Versions               | All   |

## 20204

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEFUNC  |
| Message text           | Functionality not supported at the specified version level.                 |
| Possible Cause         | A DB-Library routine that is not supported in this version has been called. |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20205

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBERESP   |
| Message text           | Response function address passed to dbresponse() must be non-NULL.                       |
| Possible Cause         | The <i>response_func</i> parameter passed to undocumented function dbresponse() is NULL. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20206

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEIVERS  |
| Message text           | Illegal version level specified.                           |
| Possible Cause         | Invalid <i>version</i> parameter passed to dbsetversion(). |
| Action/solution        | Correct application coding.                                |
| Additional information |  |
| Versions               | All  |

## 20207

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEONCE                                       |
| Message text           | Function can be called only once.              |
| Possible Cause         | dbsetversion() has been called more than once. |
| Action/solution        | Correct application coding.                    |
| Additional information |  |
| Versions               | All  |

## 20208

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBERPNULL  |
| Message text           | value parameter for dbprcparam() can be NULL, only if the datalen parameter is 0. |
| Possible Cause         | Parameters to dbrcparam() are not in agreement.                                   |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20209

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBERPTXTIM  |
| Message text           | RPC parameters cannot be of type Text/Image.   |
| Possible Cause         | The <i>type</i> parameter passed to <code>dbrpcparam()</code> cannot be SYBTEXT or SYBIMAGE. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20210

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBENEG   |
| Message text           | Negotiated login attempt failed.  |
| Possible Cause         | Failure to perform a secure login to the server.                              |
| Action/solution        | Check security credentials and security settings provided by the application. |
| Additional information |   |
| Versions               | All   |

## 20211

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBELBLEN  |
| Message text           | Security labels should be less than 256 characters long.   |
| Possible Cause         | Label values passed to undocumented routine <code>dbsetsecurity()</code> exceed <i>DB_MAX_LABELLEN</i> . |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20212

|                        |                                       |
|------------------------|---------------------------------------|
| Symbolic constant      | SYBEUMSG                              |
| Message text           | Unknown message-id in MSG datastream. |
| Possible Cause         | Internal TDS error.                   |
| Action/solution        | Contact Sybase Technical Support.     |
| Additional information |                                       |
| Versions               | All                                   |

## 20213

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECAPTYP   |
| Message text           | Unexpected capability type in CAPABILITY datastream. |
| Possible Cause         | Internal TDS error.                                  |
| Action/solution        | Contact Sybase Technical Support.                    |
| Additional information |  |
| Versions               | All  |

## 20214

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBNUM  |
| Message text           | Bad numbytes parameter passed to dbstrcpy().  |
| Possible Cause         | An invalid value for the <i>numbytes</i> parameter has been passed to the dbstrcpy() routine. |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20215

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBBL   |
| Message text           | Bad bindlen parameter passed to dbsetnull().  |
| Possible Cause         | A negative value has been passed in <i>bindlen</i> parameter of the DB-Library routine dbsetnull(). |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20216

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBPREC   |
| Message text           | Illegal precision specified.  |
| Possible Cause         | The precision specified in the DBTYPEINFO structure for a numeric or decimal column is invalid. |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20217

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBSCALE  |
| Message text           | Illegal scale specified.  |
| Possible Cause         | The scale specified in the DBTYPEINFO structure for a numeric or decimal column is invalid. |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20218

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECDOMAIN  |
| Message text           | Source field value is not within the domain of legal values.                           |
| Possible Cause         | The source value for a conversion using the dbconvert() DB-Library routine is invalid. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20219

|                        |                                   |
|------------------------|-----------------------------------|
| Symbolic constant      | SYBECINTERNAL                     |
| Message text           | Internal Conversion error.        |
| Possible Cause         |                                   |
| Action/solution        | Contact Sybase Technical Support. |
| Additional information |                                   |
| Versions               | All                               |

## 20220

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBTYPSTRV   |
| Message text           | Datatype is not supported by the server.                         |
| Possible Cause         | Server does not recognize this datatype for this version of TDS. |
| Action/solution        | Upgrade server.  |
| Additional information |  |
| Versions               | All  |

## 20221

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBBCSET                                      |
| Message text           | Unknown character-set encountered.              |
| Possible Cause         | Server specified an unrecognized character set. |
| Action/solution        | Contact Sybase Technical Support.               |
| Additional information |   |
| Versions               | All   |

## 20222

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEFENC   |
| Message text           | Password Encryption failed.  |
| Possible Cause         | Either the encryption handler installed by dbsechandle() or the default encryption handler failed. |
| Action/solution        | Correct application coding or Contact Sybase Technical Support.                                    |
| Additional information |  |
| Versions               | All  |

## 20223

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEFRES  |
| Message text           | Challenge-Response function failed.   |
| Possible Cause         | Either the login response handler installed by undocumented function dbresponse() or the default login response handler failed. |
| Action/solution        | Correct application coding or Contact Sybase Technical Support.   |
| Additional information |   |
| Versions               | All   |

## 20224

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEISRVPREC   |
| Message text           | Illegal precision returned by the server.  |
| Possible Cause         | The precision value of a decimal or numeric column falls outside the domain of legal precision values. |
| Action/solution        | Contact Sybase Technical Support.  |
| Additional information |  |
| Versions               | All  |

## 20225

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEISRVSCL  |
| Message text           | Illegal scale returned by the server.  |
| Possible Cause         | The scale value of a decimal or numeric column falls outside the domain of legal scale values. |
| Action/solution        | Contact Sybase Technical Support.  |
| Additional information |  |
| Versions               | All  |

## 20226

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEINUMCL  |
| Message text           | Invalid numeric column length returned by the server. |
| Possible Cause         | Illegal value sent by the server.                     |
| Action/solution        | Contact Sybase Technical Support.                     |
| Additional information |   |
| Versions               | All   |



## 20227

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEIDECCL  |
| Message text           | Invalid decimal column length returned by the server. |
| Possible Cause         | Illegal value sent by the server.                     |
| Action/solution        | Contact Sybase Technical Support.                     |
| Additional information |   |
| Versions               | All   |

## 20228

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECMTXT  |
| Message text           | bcp_moretext() may be used only when there is at least one text or image column in the server table. |
| Possible Cause         | bcp_moretext() has been called incorrectly.  |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20229

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECPPREC  |
| Message text           | Column <column>: Illegal precision value encountered. |
| Possible Cause         | Invalid precision value found in the host file.       |
| Action/solution        | Contact Sybase Technical Support.                     |
| Additional information |   |
| Versions               | All   |

## 20230

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBCBNPR   |
| Message text           | <code>bcp_bind()</code> : if <code>varaddr</code> is <code>NULL</code> , <code>prefixlen</code> must be 0 and no terminator should be specified. |
| Possible Cause         | <code>bcp_bind()</code> called incorrectly.  |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20231

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBCBNTYP  |
| Message text           | <code>bcp_bind()</code> : if <code>varaddr</code> is <code>NULL</code> and <code>varlen</code> greater than 0, the table column type must be <code>SYBTEXT</code> or <code>SYBIMAGE</code> and the program variable type must be <code>SYBTEXT</code> , <code>SYBCHAR</code> , <code>SYBIMAGE</code> or <code>SYBBINARY</code> . |
| Possible Cause         | <code>bcp_bind()</code> called incorrectly.  |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20232

|                   |  |
|-------------------|--|
| Symbolic constant | SYBEBCSNTYP  |
| Message text      | column number <colnum>: if <code>varaddr</code> is <code>NULL</code> and <code>varlen</code> greater than 0, the table column type must be <code>SYBTEXT</code> or <code>SYBIMAGE</code> and the program variable type must be <code>SYBTEXT</code> , <code>SYBCHAR</code> , <code>SYBIMAGE</code> or <code>SYBBINARY</code> . |
| Possible Cause    | <code>bcp_bind()</code> called incorrectly.  |
| Action/solution   | Correct application coding.  |

Additional information

Versions All

## 20233

Symbolic constant SYBEBCPCTYP

Message text `bcp_colfmt()`: If `table_colnum` is 0, `host_type` cannot be 0.

Possible Cause `bcp_colfmt()` called incorrectly. A `table_colnum` value of 0 means the column will not be copied.

Action/solution Correct application coding.

Additional information

Versions All

## 20234

Symbolic constant SYBEBCVLEN

Message text `varlen` should be greater than or equal to -1.

Possible Cause `bcp_bind()` or `bcp_colln()` has been called with a `varlen` value of less than -1.

Action/solution Correct application coding.

Additional information

Versions All

## 20235

Symbolic constant SYBEBCHLEN

Message text `host_colln` should be greater than or equal to -1.

Possible Cause Invalid value for `host_colln` passed to `bcp_colfmt_ps()`.

|                        |                             |
|------------------------|-----------------------------|
| Action/solution        | Correct application coding. |
| Additional information |                             |
| Versions               | All                         |

20236

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBBCBPREF   |
| Message text           | Illegal prefix length. Legal values are 0, 1, 2 or 4.            |
| Possible Cause         | Invalid value for <i>prefixlen</i> passed to <i>bcp_bind()</i> . |
| Action/solution        | Correct application coding.                                      |
| Additional information |  |
| Versions               | All  |

20237

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBBCBPREF   |
| Message text           | Illegal prefix length. Legal values are 1, 0, 1, 2 or 4.                   |
| Possible Cause         | Invalid value for <i>host_prefixlen</i> passed to <i>bcp_colfmt_ps()</i> . |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

20238

|                   |  |
|-------------------|--|
| Symbolic constant | SYBEBCITBNM  |
| Message text      | <i>bcp_init()</i> : <i>tblname</i> parameter cannot be NULL. |
| Possible Cause    | <i>bcp_init()</i> called incorrectly.                        |

Action/solution                      Correct application coding.

Additional information

Versions                              All

## 20239

Symbolic constant                      SYBEBCTIBLEN

Message text                              `bcp_init(): tblname parameter is too long.`

Possible Cause                              `bcp_init()` called incorrectly.

Action/solution                              Correct application coding.

Additional information

Versions                              All

## 20240

Symbolic constant                      SYBEBCSNDROW

Message text                              `bcp_sendrow()` may NOT be called unless all text data for the previous row has been sent using `bcp_moretext()`.

Possible Cause                              Not all text/image data has been sent.

Action/solution                              Correct application coding.

Additional information

Versions                              All

## 20241

Symbolic constant                      SYBEBPROCOL

Message text                              `bcp` protocol error: returned column count differs from the actual number of columns received.

|                        |                                   |
|------------------------|-----------------------------------|
| Possible Cause         | Internal column mismatch.         |
| Action/solution        | Contact Sybase Technical Support. |
| Additional information |                                   |
| Versions               | All                               |

## 20242

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBPRODEF   |
| Message text           | <code>bcp protocol error: expected default information and got none.</code> |
| Possible Cause         | Internal column mismatch.   |
| Action/solution        | Contact Sybase Technical Support.   |
| Additional information |   |
| Versions               | All   |

## 20243

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBPRONUMDEF  |
| Message text           | <code>bcp protocol error: expected number of defaults differs from the actual number of defaults received.</code> |
| Possible Cause         | Internal column default mismatch.   |
| Action/solution        | Contact Sybase Technical Support.   |
| Additional information |   |
| Versions               | All   |

## 20244

|                   |               |
|-------------------|---------------|
| Symbolic constant | SYBEBPRODEFID |
|-------------------|---------------|

|                        |   |
|------------------------|---|
| Message text           | <code>bcp protocol error: default column id and actual column id are not same.</code> |
| Possible Cause         | Internal column default mismatch.   |
| Action/solution        | Contact Sybase Technical Support.   |
| Additional information |   |
| Versions               | All   |

## 20245

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBPRONODEF  |
| Message text           | <code>bcp protocol error: default value received for column that does not have default.</code> |
| Possible Cause         | Internal column default mismatch.  |
| Action/solution        | Contact Sybase Technical Support.  |
| Additional information |  |
| Versions               | All  |

## 20246

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBPRODEFTYP  |
| Message text           | <code>bcp protocol error: default value datatype differs from column datatype.</code> |
| Possible Cause         | Internal column default mismatch.   |
| Action/solution        | Contact Sybase Technical Support.   |
| Additional information |   |
| Versions               | All   |

## 20247

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBPROEXTDEF   |
| Message text           | bcp protocol error: more than one row of default information received. |
| Possible Cause         | Internal column default mismatch.                                      |
| Action/solution        | Contact Sybase Technical Support.                                      |
| Additional information |  |
| Versions               | All  |

## 20248

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBPROEXTRES  |
| Message text           | bcp protocol error: unexpected set of results received. |
| Possible Cause         | Extra results sent from server.                         |
| Action/solution        | Contact Sybase Technical Support.                       |
| Additional information |   |
| Versions               | All   |

## 20249

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBPROBADDEF  |
| Message text           | bcp protocol error: illegal default column id received. |
| Possible Cause         | Internal column default mismatch.                       |
| Action/solution        | Contact Sybase Technical Support.                       |
| Additional information |   |
| Versions               | All   |



## 20250

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBPROBADTYP                               |
| Message text           | bcp protocol error: unknown column datatype. |
| Possible Cause         | Unknown datatype from server.                |
| Action/solution        | Contact Sybase Technical Support.            |
| Additional information |  |
| Versions               | All  |

## 20251

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBPROBADLEN  |
| Message text           | bcp protocol error: illegal datatype length received. |
| Possible Cause         | Illegal length received from the server.              |
| Action/solution        | Contact Sybase Technical Support.                     |
| Additional information |   |
| Versions               | All   |

## 20252

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBPROBADPREC                                       |
| Message text           | bcp protocol error: illegal precision value received. |
| Possible Cause         | Illegal precision received from server.               |
| Action/solution        | Contact Sybase Technical Support.                     |
| Additional information |   |
| Versions               | All   |

## 20253

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBPROBADSCL   |
| Message text           | <code>bcp protocol error: illegal scale value received.</code> |
| Possible Cause         | Illegal scale received from the server.                        |
| Action/solution        | Contact Sybase Technical Support.                              |
| Additional information |  |
| Versions               | All  |

## 20254

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEBADTYPE   |
| Message text           | <code>Illegal value for type parameter given to &lt;routine&gt;.</code> |
| Possible Cause         | Invalid <i>type</i> passed to dbsechandle().                            |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20255

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECRSNORES  |
| Message text           | <code>Cursor statement generated no results.</code> |
| Possible Cause         | Cursor statement returned no results.               |
| Action/solution        | No action necessary.                                |
| Additional information |   |
| Versions               | All   |

## 20256

|                        |  |
|------------------------|--|
| Message type           | Error  |
| Symbolic constant      | SYBECRSNOIND   |
| Message text           | One of the tables involved in the cursor statement does not have a unique index. |
| Possible Cause         | Inappropriate schema.  |
| Action/solution        | Correct database schema.   |
| Additional information |  |
| Versions               | All  |

## 20257

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECRSVIEW   |
| Message text           | A view cannot be joined with another table or a view in a cursor statement. |
| Possible Cause         | Cursor statement performs a join involving a view.                          |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20258

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECRSVIIND  |
| Message text           | The view used in the cursor statement does not include all the unique index columns of the underlying tables. |
| Possible Cause         | Incorrect cursor statement.   |
| Action/solution        | Correct application coding.   |
| Additional information |   |

|          |     |
|----------|-----|
| Versions | All |
|----------|-----|

## 20259

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECRSORD   |
| Message text           | Only fully keyset driven cursors can have 'order by', 'group by', or 'having' phrases. |
| Possible Cause         | Incorrect cursor statement.  |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20260

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECRSBUFR   |
| Message text           | Row buffering should not be turned on when using cursor APIs.   |
| Possible Cause         | Row buffering has been turned on with dbsetopt(...DBBUFFER...). |
| Action/solution        | Correct application coding.                                     |
| Additional information | Row buffering is incompatible with cursors.                     |
| Versions               | All   |

## 20261

|                   |   |
|-------------------|---|
| Symbolic constant | SYBECRSNOFREE   |
| Message text      | The DBNOAUTOFREE option should not be turned on when using cursor APIs. |
| Possible Cause    | dbsetopt has been turned on with dbsetopt(...DBNOAUTOFREE...).          |

|                 |                             |
|-----------------|-----------------------------|
| Action/solution | Correct application coding. |
|-----------------|-----------------------------|

|                        |  |
|------------------------|--|
| Additional information |  |
|------------------------|--|

|          |     |
|----------|-----|
| Versions | All |
|----------|-----|

## 20262

|                   |            |
|-------------------|------------|
| Symbolic constant | SYBECRSDIS |
|-------------------|------------|

|              |  |
|--------------|--|
| Message text | Cursor statement contains one of the disallowed phrases<br>'compute', 'union', 'for browse', or 'select into'. |
|--------------|--|

|                |                           |
|----------------|---------------------------|
| Possible Cause | Invalid cursor statement. |
|----------------|---------------------------|

|                 |                             |
|-----------------|-----------------------------|
| Action/solution | Correct application coding. |
|-----------------|-----------------------------|

|                        |  |
|------------------------|--|
| Additional information |  |
|------------------------|--|

|          |     |
|----------|-----|
| Versions | All |
|----------|-----|

## 20263

|                   |            |
|-------------------|------------|
| Symbolic constant | SYBECRSAGR |
|-------------------|------------|

|              |   |
|--------------|---|
| Message text | Aggregate functions are not allowed in a cursor<br>statement. |
|--------------|---|

|                |                           |
|----------------|---------------------------|
| Possible Cause | Invalid cursor statement. |
|----------------|---------------------------|

|                 |                             |
|-----------------|-----------------------------|
| Action/solution | Correct application coding. |
|-----------------|-----------------------------|

|                        |  |
|------------------------|--|
| Additional information |  |
|------------------------|--|

|          |     |
|----------|-----|
| Versions | All |
|----------|-----|

## 20264

|                   |              |
|-------------------|--------------|
| Symbolic constant | SYBECRSFRAND |
|-------------------|--------------|

|              |   |
|--------------|---|
| Message text | Fetch types RANDOM and RELATIVE can only be used within |
|--------------|---|

|                        |                                      |
|------------------------|--------------------------------------|
|                        | the keyset of keyset driven cursors. |
| Possible Cause         | dbcursorfetch() called incorrectly.  |
| Action/solution        | Correct application coding.          |
| Additional information |                                      |
| Versions               | All                                  |

## 20265

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECRSFLAST   |
| Message text           | Fetch type LAST requires fully keyset driven cursors.              |
| Possible Cause         | dbcursoropen() called with <i>scrollopt</i> other than CUR_KEYSET. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20266

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECRSBROL  |
| Message text           | Backward scrolling cannot be used in a forward scrolling cursor.     |
| Possible Cause         | Attempt to FETCH_PREV on a cursor opened with scrollopt CUR_FORWARD. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20267

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECRSFROWN  |
| Message text           | Row number to be fetched is outside the valid range.                    |
| Possible Cause         | Attempt to fetch a row before <i>firstrow</i> or after <i>lastrow</i> . |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20268

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECRSBSKEY  |
| Message text           | Keyset cannot be scrolled backward in mixed cursors with a previous fetch type. |
| Possible Cause         | Attempt to fetch a row before <i>firstrow</i> in a keyset-driven cursor.        |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20269

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECRSRO  |
| Message text           | Data locking or modifications cannot be made in a READONLY cursor.                 |
| Possible Cause         | dbcursor() called with <i>optype</i> other than CRS_REFRESH on a read-only cursor. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20270

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECRSNOCOUNT   |
| Message text           | The DBNOCOUNT option should not be turned on when doing updates or deletes with <code>dbcursor()</code> .                          |
| Possible Cause         | DBNOCOUNT option has been previously set with <code>dbsetopt()</code> . This is incompatible with cursor update/delete operations. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20271

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECRSTAB   |
| Message text           | Table name must be determined in operations involving data locking or modifications. |
| Possible Cause         | <code>dbcursor()</code> is called with an invalid <i>table</i> value.                |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20272

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECRSUPDNB   |
| Message text           | Update or insert operations cannot use bind variables when binding type is NOBIND.   |
| Possible Cause         | <code>dbcursorfetch()</code> has previously been called with vartype <i>NOBIND</i> . |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |



## 20273

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECRSNOWHERE  |
| Message text           | A WHERE clause is not allowed in a cursor update or insert. |
| Possible Cause         | dbcursor() called with inappropriate values argument.       |
| Action/solution        | Correct application coding.                                 |
| Additional information |   |
| Versions               | All   |

## 20274

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECRSSET   |
| Message text           | A SET clause is required for a cursor update or insert.      |
| Possible Cause         | dbcursor() argument <i>values</i> must contain a SET clause. |
| Action/solution        | Correct application coding.                                  |
| Additional information |  |
| Versions               | All  |

## 20275

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECRSUPDTAB  |
| Message text           | Update or insert operations using bind variables require single table cursors. |
| Possible Cause         | dbcursoropenl() stmt argument affects multiple tables.                         |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20276

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECRSNOUPD   |
| Message text           | Update or delete operation did not affect any rows.                                    |
| Possible Cause         | dbcursor() with <i>CRS_UPDATE</i> or <i>CRS_DELETE otype</i> argument affected 0 rows. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20277

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECRSINV  |
| Message text           | Invalid cursor statement.                             |
| Possible Cause         | dbcursoropen() stmt does not contain a select clause. |
| Action/solution        | Correct application coding.                           |
| Additional information |   |
| Versions               | All   |

## 20278

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECRSNOKEYS  |
| Message text           | The entire keyset must be defined for KEYSET type cursors. |
| Possible Cause         | Tables involved in a KEYSET cursor must have keys.         |
| Action/solution        | Correct application coding or database schema.             |
| Additional information |  |
| Versions               | All  |

## 20279

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECRSNOBIND  |
| Message text           | Cursor bind must be called prior to dbcursor invocation. |
| Possible Cause         | dbcursor() called with NULL <i>values</i> argument.      |
| Action/solution        | Correct application coding                               |
| Additional information |  |
| Versions               | All  |

## 20280

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECRSFTYPE   |
| Message text           | Unknown fetch type.  |
| Possible Cause         | dbcursorfetch() has been called with an unknown value for <i>fetchtype</i> . |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20282

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECRSMROWS  |
| Message text           | Multiple rows are returned, only one is expected while retrieving dbname. |
| Possible Cause         | Internal cursor initialization error.                                     |
| Action/solution        | Contact Sybase Technical Support.   |
| Additional information |   |
| Versions               | All   |

## 20283

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECRSNROWS                                |
| Message text           | No rows returned, at least one is expected. |
| Possible Cause         |   |
| Action/solution        | Contact Sybase Technical Support.           |
| Additional information | Unexpected internal error.                  |
| Versions               | All   |

## 20284

|                        |                                       |
|------------------------|---------------------------------------|
| Symbolic constant      | SYBECRSNOLEN                          |
| Message text           | No unique index found.                |
| Possible Cause         | No unique index exists for the table. |
| Action/solution        | Correct database schema.              |
| Additional information |                                       |
| Versions               | All                                   |

## 20285

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECRSNOPTCC                              |
| Message text           | No OPTCC was found.                        |
| Possible Cause         | Could not find columns with CUR_OPTCC set. |
| Action/solution        | Contact Sybase Technical Support.          |
| Additional information |  |
| Versions               | All  |

## 20286

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBECRSNORDER   |
| Message text           | The order of clauses must be 'from', 'where', and 'order by'. |
| Possible Cause         | Clauses in stmt passed to dbcursoropen() are out of order.    |
| Action/solution        | Correct application coding.                                   |
| Additional information |   |
| Versions               | All   |

## 20287

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECRSNOTABLE                         |
| Message text           | Table name is NULL.                    |
| Possible Cause         | table passed to dbcursor() is invalid. |
| Action/solution        | Correct application coding.            |
| Additional information |  |
| Versions               | All                                    |

## 20288

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECRSNUNIQUE                                       |
| Message text           | No unique keys associated with this view.            |
| Possible Cause         | No keys associated with tables underlying this view. |
| Action/solution        | Correct database schema.                             |
| Additional information |  |
| Versions               | All  |

## 20289

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBECRSVAR   |
| Message text           | There is no valid address associated with this bind.                           |
| Possible Cause         | dbcursorbind() called with <i>vartype NOBIND</i> and invalid <i>pvaraddr</i> . |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20290

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBENOVALUE   |
| Message text           | Security labels require both a name and a value.  |
| Possible Cause         | Security label handler has set either a <i>namelen</i> or <i>valuelen</i> to a value $\leq 0$ . |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20291

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEVOIDRET   |
| Message text           | Return parameter cannot be of the type SYBVOID.                         |
| Possible Cause         | dbrpcparam() attempted to define a return parameter as <i>SYBVOID</i> . |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20292

|                        |                                   |
|------------------------|-----------------------------------|
| Symbolic constant      | SYBECLOSEIN                       |
| Message text           | Unable to close interfaces file.  |
| Possible Cause         | Internal close failure.           |
| Action/solution        | Contact Sybase Technical Support. |
| Additional information |                                   |
| Versions               | All                               |

## 20293

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBOOL   |
| Message text           | Value of a boolean parameter should be either TRUE or FALSE.   |
| Possible Cause         | bcpl_options() has been called with option <i>BCPLABELED</i> and <i>*value</i> is not set to either TRUE or FALSE. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20294

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBCOPT  |
| Message text           | The <option> option cannot be called while a bulk copy operation is in progress.                     |
| Possible Cause         | bcpl_options() with option <i>BCPLABELED</i> has been called when a bulk-copy operation is progress. |
| Action/solution        | Correct application coding.  |
| Additional information |  |
| Versions               | All  |

## 20295

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEERRLABEL  |
| Message text           | An illegal value was returned from the security label handler.  |
| Possible Cause         | Security label handler returned a value other than <i>DMORELABEL</i> , <i>DBENDLABEL</i> , or <i>DBERRLABEL</i> . |
| Action/solution        | Correct application coding.   |
| Additional information |   |
| Versions               | All   |

## 20296

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEATTNACK  |
| Message text           | Timed out waiting for server to acknowledge attention. |
| Possible Cause         | The server has not responded to a dbcancel() request.  |
| Action/solution        | Contact Sybase Technical Support.                      |
| Additional information |  |
| Versions               | All  |

## 20297

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBEBBFL                                 |
| Message text           | Batch failed in bulk-copy to the server. |
| Possible Cause         | Server reported an error in bulk-copy.   |
| Action/solution        | Contact Sybase Technical Support.        |
| Additional information |  |
| Versions               | All                                      |



## 20298

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEDCL   |
| Message text           | A directory control layer (DCL) error occurred. |
| Possible Cause         | Error reading directory service.                |
| Action/solution        | Contact Sybase Technical Support.               |
| Additional information |   |
| Versions               | All   |

## 20299

|                        |                                   |
|------------------------|-----------------------------------|
| Symbolic constant      | SYBECS                            |
| Message text           | A CS Context error occurred.      |
| Possible Cause         | Obsolete                          |
| Action/solution        | Contact Sybase Technical Support. |
| Additional information |                                   |
| Versions               | None.                             |

## 20300

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEVERENV  |
| Message text           | An invalid value was used for SYBOCS_DBVERSION.                 |
| Possible Cause         | dbsetversion() has been called with an invalid <i>version</i> . |
| Action/solution        | Correct application coding.                                     |
| Additional information |   |
| Versions               | All   |

## 20301

|                        |  |
|------------------------|--|
| Symbolic constant      | SYBCOPNOV  |
| Message text           | <code>dbcursoropen()</code> : The multiplication of <code>scrollopt</code> and <code>nrows</code> results in overflow. |
| Possible Cause         |  |
| Action/solution        | Contact Sybase Technical Support.  |
| Additional information | Internal cursor error.   |
| Versions               | 15.7 and later   |

## 20302

|                        |   |
|------------------------|---|
| Symbolic constant      | SYBEINTOVFL   |
| Message text           | <code>DB-LIBRARY</code> internal error: The arithmetic operation results in integer overflow. |
| Possible Cause         | A buffer is not large enough for character set translation.                                   |
| Action/solution        | Contact Sybase Technical Support.   |
| Additional information |   |
| Versions               | 15.7 and later  |

# Index

## A

- abort\_xact 468
- Adaptive Server
  - updating among multiple 453
- aggregate operators
  - returning for a compute column 67
- application names
  - setting in LOGINREC 317
- applications
  - DB-Library/C 6, 7, 12
  - gateway 251, 303
- arithmetic exceptions 407
- and Secure Adaptive Server 447
- sending data from program variables 448
- sending text/image values 444
- setting LOGINREC for 450
- setting number of columns in host file 435
- setting options for 447
- specifying host file format 426, 429
- writing format definitions to a file 451

- bcp\_batch 421
- bcp\_bind 422, 426
- bcp\_colfmt 426, 429
- bcp\_colfmt\_ps 429, 433
- bcp\_colln 434
- bcp\_colptr 435
- bcp\_columns 435, 436
  - and bcp\_bind 424
- bcp\_control 436, 439
- bcp\_done 439
  - and bcp\_bind 426
- bcp\_exec 439, 441
- bcp\_getl 441
- bcp\_init 441, 444
- bcp\_moretext 444, 447
- bcp\_options 447
- bcp\_readfmt 448
- bcp\_sendrow 448, 449
- BCP\_SETL 450
- bcp\_setxlate 450, 451
- bcp\_writfmt 451, 452
- binary data
  - reading page of 247
  - writing page of to the server 381
- bind result column to program variable 72, 77
- browse mode 26, 28
  - and DBPROCESS 27
  - determining number of tables involved 365
  - determining whether regular column source is updatable 93
  - identifying browsable tables 363
- buffers

## B

- batches
  - command. See Command batches 84
- bcp
  - BCPLABELED option 447
  - binding data 422
  - changing allowable number of errors 437
  - changing default data formats 436
  - changing first row to copy 437
  - changing last row to copy 437
  - changing number of rows to copy 437
  - changing program variable data address 435
  - changing program variable data length 434
  - character set translations for 450
  - copying multiple files 448, 451
  - default data formats 443
  - enabling 450
  - ending bulk copy from program variables 439
  - executing 439
  - host file format 426, 429
  - initializing 441
  - overriding default data formats 426, 429, 435
  - reading format definitions 448
  - saving preceding rows in Adaptive Server 421

- command. See Command buffers 6
- determining size for results 343
- placing query results header in 344
- row. See Row buffers 6
- build\_xact\_string 468, 469
- bulk copy 417, 419
- bylist 84
  - returning 83

**C**

- chained transactions 407
- character set
  - returning client 157
  - returning server 305
  - setting 318
  - setting default 310
  - translation 87
- character set translations
  - freeing tables 153
  - loading tables 175
  - specifying for bcp 450
  - for strings 387
  - tables 154
- character strings
  - and quotation marks 297
  - translating from one character set to another 387
- characters
  - getting from command buffer 157
- Client/server
  - architecture 1, 2
- Client-Library
  - definition 4
- clients
  - types of 2
- close\_commit 469, 470
- columns
  - compute. See Compute columns 49
  - regular. See Regular columns 72
  - returning ID of in order by clause 233
  - returning number in order by clause 227
- command batches 15
  - canceling current 84
  - determining whether more results to process 210
  - sending to the server 347, 354

- setting results for next command 275
- and switching databases 86
- verifying correctness of 349
- command buffers 6, 15
  - adding text to 91, 149
  - checking for Transact-SQL constructs 162
  - clearing 154
  - copying portions of 359
  - getting characters 157
  - and message handling 214
  - returning character length of 361
  - setting no clear option 92
- commands
  - canceling entire batch 85
  - determining whether it can return rows 92
  - determining whether it returned rows 289
  - determining whether more to process 210
  - getting stored procedures status number 285
  - processing 15, 16
  - returning number of current 113
  - returning number of rows affected by 112
  - setting up results for next 275
- commit\_xact 470
- comparing
  - datetime values 129
- compute clauses
  - returning number in results 227
- compute columns
  - associating with indicator variables 70
  - binding to program variables 54, 59
  - getting data 49
  - order returned 50
  - returning data length of 52
  - returning maximum data length of 66
  - returning number in row 225
  - returning select-list id 65
  - returning server datatype for 68
  - returning type of row aggregate 67
  - returning user-defined datatypes for 69
  - summing or averaging 50
- compute rows 16
  - determining 217
  - getting data for a column 50
  - reading next 217
  - returning bylist for 83
  - returning number of columns 225

creating a notification procedure 219, 222  
 creating a registered procedure 219, 222

## CS-Library

definition 4

## cursor

binding 117  
 closing 119  
 fetching against 121  
 opening 124  
 retrieving column information for 120  
 retrieving information about 123  
 updating 115

# D

## data

getting user-allocated 167  
 reading binary 247  
 reading server (UNIX) 170  
 saving user-allocated 336  
 writing binary 381  
 writing to the server (UNIX) 172

## databases

determining whether changed 86  
 multi-user updates 26  
 reading pages 247  
 returning name of current 216  
 updating 26, 28  
 updating on multiple servers 453  
 using specified 376

## datatypes

Adaptive Server 11  
 binding compute columns to 54, 59  
 binding regular columns to 72, 77  
 compute columns 68  
 conversions supported 104, 110  
 conversions supported by dbaltbind 55  
 conversions supported by dbaltbind\_ps 61  
 conversions supported by dbbind 73  
 conversions supported by dbbind\_ps 78  
 converting 102, 106  
 converting to same 105, 112  
 DB-Library 412, 416  
 DB-Library/C 11  
 determining supported conversions 379

getting precision and scale for regular column 99  
 returning for compute columns 68, 69  
 returning for regular column 98  
 server 103, 109  
 server, list of 412  
 user-defined, for compute columns 69  
 user-defined, for regular columns 100

## date formats

input 408

DATEFIRST option 407

DATEFORMAT option of set command 408

## dates

converting parts to character strings 135  
 converting to character format 131  
 converting values into usable format 133  
 determining month name in specified language 209  
 parts of 131  
 returning name of day in specified language 142  
 returning order for specified language 138  
 returning parts as numeric values 139  
 symbols recognized by DB-Library 131

datetime routines 34, 35

## datetime values

comparing 129

## days

returning name of in specified language 142

db12hour 48, 49

dbadata 49, 52

as alternate to dbaltbind 58, 65

dbadlen 52, 54

dbaltbind 54, 59

as alternate to dbadata 51

dbaltbind\_ps 59, 65

dbaltcolid 65, 66

dbaltlen 66, 67

dbaltop 67, 68

dbalttype 68, 69

dbaltutype 69, 70

dbanullbind 70, 71

DBARITHABORT option 407

DBARITHIGNORE option 407

DBAUTH option 407

dbbind 72, 77

as alternate to dbdata 129

dbbind\_ps 77, 82

- DBBUFFER option 407
  - and DBFIRSTROW 153
  - and dbgetrow 166
  - and DBLASTROW 175
  - and reading result rows 218
- dbbufsize 82, 83
- dbbylist 83, 84
- dbcancel 84, 85
- dbcquery 85, 86
- DBCHAINXACTS option 407
- dbchange 86, 87
- dbcharsetconv 87
- dbclose 88
- dbclrbuf 88, 89
- dbclropt 89, 91
- dbcmd 91, 92
- DBCMDROW 92, 93
- dbcolbrowse 93, 94
- dbcollen 94, 95
- dbcolname 95, 96
  - and returning bylist 83
- dbcolsource 97, 98
- dbcoltype 98, 99
- dbcoltypeinfo 99, 100
- dbcolutype 100, 102
- dbconvert 102, 106
- dbconvert\_ps 106, 112
- DBCOUNT 112, 113
- DBCURCMD 113, 114
- DBCURROW 114, 115
- dbcursor 115, 116
- dbcursorbind 116, 119
- dbcursorcloses 119
- dbcursorcolinfo 119, 120
- dbcursorfetch 120, 123
- dbcursorinfo 123, 124
- dbcursoropen 124, 128
- dbdata 128, 129
  - as alternate to dbbind 76, 82
- dbdate4cmp 129, 130
- dbdate4zero 130, 131
- dbdatechar 131, 132
- dbdatecmp 132, 133
- dbdatecrack 133, 135
- dbdatename 135, 138
- dbdateorder 138, 139
- dbdatepart 139, 140
- DBDATERECD structure 133
- DBDATETIME structure 134
  - converting date parts to character strings 135
  - converting integer component to character format 131
  - converting values into usable format 133
  - returning parts as numeric values 139
- dbdatezero 140, 141
- dbdatlen 141, 142
- dbdayname 142, 143
- DBDEAD 143, 144
- dberrhandle 144, 148
- dbexit 148, 149
- dbfcmd 149, 152
- DBFIRSTROW 152, 153
  - and dbgetrow 166
- dbfree\_xlate 153, 154
- dbfreebuf 154, 155
- dbfreeequal 155
- dbfreesort 155, 156
- dbgetchar 157
- dbgetcharset 157, 158
- dbgetloginfo 158, 160
- dbgetlusername 160, 161
- dbgetmaxprocs 161, 162
- dbgetnatlang 162
- dbgetoff 162, 164
- dbgetpacket 164, 165
- dbgetrow 165, 166
- DBGETTIME 167
- dbgetuserdata 167, 168
- dbhasretstat 168, 170
- dbinit 170
  - and dbexit 149
- DBIORDER (UNIX) 170, 171
- DBIOWDESC (UNIX) 172
- DBISAVAIL 173
- dbisopt 173, 174
- DBLASTROW 174, 175
- DB-Library 4
  - determining version in use 378
  - initializing 170
- dbload\_xlate 175, 176
  - and freeing translation tables 154
- dbloadsort 176, 177

- dblogin 177, 179
- dbloginfree 179
- dbmny4add 179, 180
- dbmny4cmp 180, 181
- dbmny4copy 181, 182
- dbmny4divide 182, 183
- dbmny4minus 183, 184
- dbmny4mul 184, 185
- dbmny4sub 185, 186
- dbmny4zero 186, 187
- dbmnyadd 187, 188
- dbmnycmp 188, 189
- dbmnycopy 189, 190
- dbmnydec 190, 191
- dbmnydivide 191, 192
- dbmnydown 192, 193
- dbmnyinc 194
- dbmnyinit 194, 196
- dbmnymaxneg 196, 197
- dbmnymaxpos 197
- dbmnyminus 198
- dbmnymul 199
- dbmnyndigit 200, 206
- dbmnyyscale 206, 207
- dbmnysub 208
- dbmnyzero 209
- dbmonthname 209, 210
- DBMORECMDS 210, 211
- dbmoretext 211, 212
- dbmsghandle 212, 216
  - and dberrhandle 147
- dbname 216, 217
- DBNATLANG option 408
- dbnextrow 217, 219
  - and DBROWS 289
  - and DBROWTYPE 289
- DBNOAUTOFREE option
  - and dbfcmd 151
  - and dbfreebuf 154
- DBNOCOUNT option
  - and DBCOUNT 113
- dbnpcreate 219, 221
- dbnpdefine 222, 223
- dbnullbind 224
- dbnumalts 225
- dbnumcols 225, 226
- dbnumcompute 227
- DBNUMORDERS 227, 228
- dbnumrets 228, 229
- DBOFFSET option
  - and dbgetoff 163
- dbopen 229, 233
  - getting a LOGINREC 177
  - setting login response time 325
- dbordercol 233, 234
- DBPARSEONLY option 409
- dbpoll 234, 239
- DBPRCOLSEP option
  - and dbsprlrow 342
- dbprhead 239, 240
- DBPRLINELEN option 410
- DBPRLINESEP option 410
- DBPROCESS structure 6
  - allocating 229
  - closing a 88
  - closing all 148
  - de-allocating a 88
  - de-allocating all 148
  - determining current limit available 161
  - determining whether available 173
  - determining whether dead 143
  - getting client character set from 157
  - getting national language from 162
  - getting server process ID 340
  - getting user-allocated data 167
  - initializing 229
  - marking available 306
  - multiple 6
  - saving user-allocated data 336
  - setting maximum number available 329
  - sharing single 173
  - and two-phase commit service 455
- DBPRPAD option 410
  - and dbsprlrow 342
- dbprrow 240, 241
- dbprtype 241, 242
- dbqual 242, 245
  - freeing allocated memory 155
- DBRBUF (UNIX) 246
- dbreadpage 247
- dbreadtext 248, 250
- dbrecftos 250

- dbrecvpassthru 251, 253
- dbregdrop 253, 254
- dbregexec 254, 256
- dbreghandle 256, 260
- dbreginit 260, 262
- dbreglist 262, 263
- dbregnowatch 263, 265
- dbregparam 265, 269
- dbregwatch 269, 274
- dbregwatchlist 274, 275
- dbresults 275, 278
- dbretdata 278, 281
- dbretlen 282, 283
- dbretname 283, 285
- dbretstatus 285, 286
- dbrettype 287, 289
- DBROWCOUNT option 410
- DBROWS 289
- DBROWTYPE 289, 290
- dbrpcinit 290, 292
- dbrpcparam 292, 294
  - and return parameter values 228
- dbrpcsend 294, 295
- dbrpwclr 295, 296
- dbrpwset 296, 297
- dbsafestr 297, 298
- dbsechandle 299, 302
- dbsendpassthru 303, 305
- dbservcharset 305
- dbsetavail 306
- dbsetbusy 306, 309
- dbsetconnect 309
- dbsetdefcharset 310, 311
- dbsetdeflang 311, 312
- dbsetidle 312, 313
- dbsetifile 313, 314
- dbsetinterrupt 314, 317
  - and canceling result rows 86
- DBSETLAPP 317, 318
- DBSETLCHARSET 318, 319
- DBSETLENCRYPT 319, 320
- DBSETLHOST 320, 321
- DBSETLMUTUALAUTH 321
- DBSETLNATLANG 322
- DBSETLNETWOKRAUTH 322
- dbsetloginfo 323, 325
- dbsetlogintime 325, 326
- DBSETLPACKET 326, 327
- DBSETLPWD 327, 328
- DBSETLSERVERPRINCIPAL 328
- DBSETLUSER 329
- dbsetmaxprocs 329, 330
- dbsetnull 330, 332
- dbsetopt 332, 334
- dbsetrow 334, 335
- dbsettime 336
- dbsetuserdata 336, 339
- dbsetversion 339, 340
- DBSHOWPLAN option 410
- dbspid 340, 341
- dbsprlrow 341, 343
  - and dbsprlrowlen 344
- dbsprlrowlen 343, 344
- dbsprhead 344, 346
  - and dbsprlrowlen 344
  - and dbsprline 346
- dbsprline 346, 347
  - and dbsprlrowlen 344
- dbsqlxec 347, 349
- dbsqlok 349, 354
- dbsqlsend 354, 355
- DBSTAT option 410
- DBSTORPROCID option 410
- dbstrbuild 355, 357
- dbstrcmp 358, 359
  - and dbstrsort 363
- dbstrcpy 359, 361
- dbstrlen 361, 362
- dbstrsort 362, 363
- dbtabbrowse 363, 364
- dbtabcount 364, 365
- dbtabname 365, 366
  - and dbtabcount 365
- dbtabsource 366, 367
- DBTDS 368
- dbtextsize 368, 369
- DBTEXTSIZE option 411
- dbtsnewlen 369, 370
- dbtsnewval 370, 371
- dbtsput 371, 372
- dbtxptr 372, 373
- dbtxtimestamp 374



- dbtxtsnewval 375
- dbtxtsput 375, 376
- dbuse 376, 377
- dbvarylen 377, 378
- dbversion 378
- dbwillconvert 379, 380
- dbwritepage 381
- dbwritetext 382, 387
- dbxlate 387, 389
- deadlock
  - handling 168, 337
- debugging
  - and dbprhead 239
  - and dbprrow 240
  - and dbsprlrow 341
  - and dbsprlrowlen 344
  - and dbsprhead 345
  - and dbsprline 347
  - recording SQL text sent to the server 250
  - for two-phase commit service 468
- decimal datatype
  - getting precision and scale for regular column 99
- default character set
  - setting for an application 310
- default language
  - setting for an application 311
- defining a notification procedure 222
- defining a registered procedure 222
- distributed transactions. *See* Two-phase commit service 453
- dropping a registered procedure 253

## E

- embedded SQL
  - comparing Client-Library to 5
- encrypted passwords 319
- encryption handler
  - installing 299
- error handling 22
  - and converting datatypes 103, 108
  - and DBDEAD 144
  - installing a user-function 144, 148
  - list of errors 389
  - translating messages from one language to another

- 355
- uninstalling handler 147
- error severity values 9
- errors 389, 406
  - DB-Library 389
- executing a registered procedure 254, 260
- exit values 9

## F

- file descriptors (UNIX)
  - access to 170, 172
- files
  - header 9
- functions
  - user-supplied to handle interrupts 314
  - user-supplied, indicating DB-Library is finished
    - reading from the server 312
  - user-supplied, indicating server access 306

## G

- gateway applications 33, 251, 303
- getting
  - the client character set 157
  - the national language 162
  - the server character set 305

## H

- handler
  - error 22
  - message 22
  - notification 256
- header files 9
- host names
  - setting in LOGINREC 320

## I

- image values
  - bulk copying parts 444

- bytes left of 368
- limiting size of 411
- reading parts of 248
- and text pointers 373
- and text timestamps 374
- updating 211, 382
- include files 9
- input streams
  - and checking for unread bytes in network buffer (UNIX) 246
  - responding to multiple (UNIX) 171
  - utilizing multiple (UNIX) 172, 355
- interfaces file
  - and dbopen 230
  - specifying name and location 313
- interrupt handling 314

## L

- languages
  - getting name from DBPROCESS 162
  - setting default 311
  - setting name in LOGINREC 322
  - setting national 408
- line length
  - specifying for rows 410
- listing registered procedures 262
- listing requested registered procedure notifications 274
- logging into the server 229
- login record. See LOGINREC structure 6
- LOGINREC structure 6
  - adding remote passwords 296
  - allocating 177
  - clearing all remote passwords 295
  - freeing 179
  - packet size field 164, 326
  - setting application name in 317
  - setting client character set in 318
  - setting for bcp 450
  - setting host name in 320
  - setting password in 327
  - setting user language name in 322
  - setting username in 329
- logins, secure 299

## M

- message handling 22
  - and dberrhandle 147
  - and dbreadpage 247
  - and deadlock 337
  - installing a user function 212, 216
  - uninstalling handler 214
- MIT Kerberos 36
- money routines 34, 35
- months
  - determining name in specified language 209
- multiple input streams 234

## N

- network buffers
  - determining whether unread bytes (UNIX) 246
  - polling 234
- network connections
  - closing 88
  - specifying interfaces file 313
- notification handler 256
- notification procedure
  - creating 219, 222
  - defining 222
- notification request
  - canceling 263
  - listing 274
- notifications
  - listing registered procedures 274
  - registered procedure 256
- null values
  - binding 330
  - default 331
  - defining 330
- numeric datatype
  - getting precision and scale for regular column 99

## O

- offsets
  - types of 163
- open\_commit 470, 471
- options 407, 412

- checking status of 173
- clearing 89
- DB-Library 407
- parameter values of 411
- setting 332
- order by clauses
  - returning column ID in 233
  - returning number of columns in 227
- output streams
  - utilizing multiple (UNIX) 172, 355

## P

- packet size
  - TDS 164, 326
- padding
  - specifying characters to use 410
- parameters
  - registered procedure 265
- passthrough operation 251, 303
- passwords
  - remote, adding 296
  - remote, clearing 295
  - setting server 327
- polling the network buffer 234
- process ID
  - getting 340
- processing plan
  - generating description of 410
- programming
  - DB-Library/C 6, 12

## Q

- queries
  - aborting during arithmetic exceptions 407
  - ignoring arithmetic exceptions 407
- quotation marks
  - and character strings 297

## R

- registered procedure 31, 33

- canceling notification request 263
- creating 219, 222
- defining 222
- dropping 253
- example 32
- executing 254, 260
- handler routine 256
- listing currently defined 262
- listing requested notifications 274
- notifications 234, 254, 256
- parameters 265
- requesting notifications 269
- routines 33
- uses of 31
- regular columns
  - associating indicator variables with 224
  - binding to program variables 72, 77
  - determining number of in results 225
  - determining whether data length can vary 377
  - determining whether source column is updatable
    - with browse mode 93
  - getting data 128
  - getting precision and scale with dbcoltypeinfo 99
  - returning data length of 141
  - returning datatypes for 98
  - returning maximum data length of 94
  - returning name of 95
  - returning name of source column 97
  - returning user-defined datatypes for 100
- regular rows 16
  - determining 217
  - limiting number to return 410
  - reading next 217
- remote procedure calls 30, 291
  - adding parameters to 292
  - adding passwords for 296
  - advantages of 291
  - clearing passwords 295
  - determining number of return parameter values
    - 228
  - determining whether status number was generated
    - 168
  - getting datatype of return parameter value 287
  - getting length of return parameter value 282
  - getting name of return parameter value 283
  - getting return parameter values 278

- getting status number 285
- initializing 290
- processing 30, 278
- signaling end of 294
- remove\_xact 471, 472
- requesting a registered procedure notification 269
- result columns
  - compute. See Compute columns 49
  - regular. See Regular columns 72
  - returning name and number of source table 366
- result rows 16
  - buffering 407
  - canceling 85
  - compute 16
  - dropping from buffer 88
  - placing header in buffer 344
  - printing 240
  - printing column headings of 239
  - processing 16, 22
  - putting one in buffer 341
  - reading next 217
  - regular 16
- results
  - setting up for next query 275
- return parameter values 228
  - determining number of 228
  - getting 278
  - getting datatype of 287
  - getting length of 282
  - getting parameter name 283
- returning TDS packet size 164
- returning the client character set 157
- returning the national language 162
- returning the server character set 305
- routines 12, 36
  - browse mode 27
  - command processing 15
  - error handling 22
  - image handling 28
  - information retrieval 24
  - initialization 13
  - message handling 22
  - process control 30
  - registered procedure 33
  - remote procedure call 30
  - results processing 16

- TDS 33
  - text handling 28
  - two-phase commit service 36
- row aggregates
  - returning for a compute column 67
- row buffers 166, 335
  - clearing 88
  - reading specified rows 165
  - returning number of first row 152
  - returning number of last row 174
- rows
  - buffering 407
  - compute 16
  - determining type 217
  - determining whether returned 289
  - determining whether returned by command 92
  - dropping from buffer 88
  - limiting number to return 410
  - printing 240
  - printing column headings of 239
  - reading next 217
  - reading specified in buffer 165
  - regular 17
  - result. See Result rows 16
  - returning number affected by a command 112
  - returning number of current 114
  - returning number of first in buffer 152
  - returning number of last in buffer 174
  - returning type of 289
  - specifying line length 410
  - specifying separator characters 410
  - updating current in browsable table 242

## S

- sample programs
  - DB-Library/C 7
- scan\_xact 472
- secure Adaptive Server
  - and bcp 447
  - routines for 35
- secure logins
  - installing user function for 299
- security label handler
  - installing 301

- separator characters
  - specifying for rows 410
- server 85, 167
  - communicating with 6
  - converting token values 241
  - datatypes 103, 109
  - logging into 229
  - reading data from (UNIX) 170
  - recording SQL text sent to 250
  - sending text/image values to 211
  - setting response time 325
  - setting user passwords 327
  - types 2
  - writing data to (UNIX) 172
- servers
  - multiple 6
- setting TDS packet size 326
- setting the client character set 318
- sort orders 156
  - comparing two character strings 358
  - determining order of two character strings 362
  - freeing 155
  - loading 176
- sprintf function 149
- SQL text
  - recording 250
- start\_xact 472, 473
- stat\_xact 473, 474
- statistics
  - performance, determining when returned 410
- status numbers
  - for current command 285
  - determining whether generated 168
- stored procedures
  - calling remotely 30
  - determining number of return parameter values 228
  - return parameter values, getting 278
  - return parameter values, getting datatype of 287
  - return parameter values, getting length of 282
  - return parameter values, getting parameter name of 283
  - returning status number 285
  - sending ids of 410
  - and status numbers 168
- sybdb.h header file 9, 145, 163

- and DB-Library options 407
- and error handling 389
- syberror.h header file 9, 147
  - and error severities 389
- SYBESMSG
  - and error handling 147
- sybfront.h header file 9
  - and interrupt handling 315
- syntax
  - checking 409

## T

- tables
  - determining names of 365
  - identifying browsable 363
  - returning name and number associated with result columns 366
  - returning name of 365
  - returning number involved in a select query 364
  - server work 365
- Tabular Data Stream
  - protocol 368
  - routines 33, 164, 251, 303, 326
- TDS
  - determining packet size 164
  - passthrough operation 251, 303
  - routines 33
  - setting packet size 326
- TDS buffer
  - polling 234
- TDS packet
  - receiving 251
  - sending 303
- text and image data 164, 326
- text pointers 373
  - returning value of 372
- text timestamps 374
  - putting new value into DBPROCESS 375
  - returning value of 374
  - returning value of after update 375
- text values
  - bulk copying parts 444
  - bytes left of 368
  - limiting size of 411

- reading parts of 248
  - and text pointers 373
  - and text timestamps 374
  - updating 211, 382
- text/image data
  - updating 212
- time
  - amount DB-Library waits for a server response 167
  - determining when to return status 410
  - determining whether 12 or 24-hour 48
  - setting length DB-Library waits for server response 336
  - setting server login response 325
- timestamp columns 26
  - putting new value in DBPROCESS 371
  - returning length of after update 369
  - returning value of after update 370
  - and updating rows 243
- token values
  - converting to readable strings 241
- transactions
  - distributed. See Two-phase commit service 453
- Transact-SQL commands
  - and DBPROCESS 6
- translation tables
  - freeing 153
  - loading 175
- two-phase commit service 453, 467
  - building names for recovery purposes 468
  - closing connections 469
  - and DBPROCESS 455
  - debugging 468
  - decrementing site count 471
  - diagnostic routines 472, 473
  - and interfaces file 457
  - marking transactions as aborted 468
  - marking transactions as committed 470
  - opening connections 470
  - printing record of distributed transactions 472
  - returning status of a distributed transaction 473
  - routines for 36
  - starting a distributed transaction 472
- typedefs
  - DB-Library/C 11
  - DB-Library/C, list of 413

## U

- updating databases 26, 28
  - on multiple servers 453
  - multi-user situations 26
  - and text/image data 212
- user names
  - setting 329
- user-defined datatypes
  - returning for a compute column 69
  - returning for regular columns 100
- user-supplied data
  - retrieving for a DBPROCESS 167
  - saving in a DBPROCESS 336
- user-supplied functions
  - calling to handle interrupts 314
  - indicating DB-Library is finished reading from the server 312
  - indicating server access 306

## V

- versions
  - DB-Library, determining which 378

## W

- where clauses
  - for use in updating a browsable table 242