PUBLIC

SDK for SAP Adaptive Server Enterprise 16.0 SP03

Document Version: 1.0 – 2020-03-04

DB-Library/C Reference Manual

THE BEST RUN

Content

1 DB-Library. 18

1.1 Client/Server Architecture. 18

1.2 Types of Clients. 19

1.3 Types of Servers. 19

1.4 Open Client and SAP Open Server solution Products. 19

Open Client. 20

SAP Open Server Solution. 20

Open Client Libraries. 20

What is in DB-Library/C?. 21

Comparing the Library approach to Embedded SQL. 21

1.5 Data Structures for Communicating with Servers 22

1.6 Writing a DB-Library/C Program. 22

DB-Library/C Datatypes 26

1.7 DB-Library/C Routines. 26

Initialization. 28

Command Processing. 31

Results Processing. 34

Message and Error Handling. 40

Information Retrieval. 41

Browse Mode. 44

Text and Image Handling. 46

Datatype Conversion. 48

Process Control Flow. 48

Remote Procedure Call 49

Registered Procedure Call. 50

Gateway Passthrough Routines. 51

Datetime and Money. 52

Cleanup. 53

Secure Support. 54

Miscellaneous Routines. 54

Two-Phase Commit Service Special Library. 55

1.8 MIT Kerberos on DB-Library. 55

Installing MIT-Kerberos on DB-Library. 56

1.9 Sample Programs. 56

2 Routines. 57

2.1 db12hour. 62

2.2 dbadata. 64
2.3 dbadlen. 66
2.4 dbaltbind. 68
2.5 dbaltbind_ps. 74
2.6 dbaltcolid. 79
2.7 dbaltlen. 81
2.8 dbaltop. 82
2.9 dbalttype. 84
2.10 dbaltutype. 85
2.11 dbanullbind. 87
2.12 dbbind. 88
2.13 dbbind_ps. 93
2.14 dbbufsize. 98
2.15 dbbylist. 99
2.16 dbcancel. 101
2.17 dbcanquery. 102
2.18 dbchange. 104
2.19 dbcharsetconv. 105
2.20 dbclose. 106
2.21 dbclrbuf. 107
2.22 dbclropt. 108
2.23 dbcmd. 110
Usage for dbcmd. 111
2.24 DBCMDROW. 112
2.25 dbcolbrowse. 113
2.26 dbcollen. 115
2.27 dbcolname. 116
2.28 dbcolsource. 118
2.29 dbcoltype. 119
2.30 dbcoltypeinfo. 121
2.31 dbcolutype. 123
2.32 dbconvert. 124
2.33 dbconvert_ps. 128
2.34 DBCOUNT. 134
2.35 DBCURCMD. 135
2.36 DBCURROW. 136
2.37 dbcursor. 137
2.38 dbcursorbind. 139
2.39 dbcursorclose. 142
2.40 dbcursorcolinfo. 143
2.41 dbcursorfetch. 144

2.42 dbcursorinfo. . . . 147
2.43 dbcursoropen. . . . 148
2.44 dbdata. . . . 151
2.45 dbdate4cmp. . . . 153
2.46 dbdate4zero. . . . 154
2.47 dbdatechar. . . . 155
2.48 dbdatecmp. . . . 157
2.49 dbdatecrack. . . . 158
2.50 dbdatename. . . . 161
2.51 dbdateorder. . . . 163
2.52 dbdatepart. . . . 165
2.53 dbdatezero. . . . 167
2.54 dbdatlen. . . . 168
2.55 dbdayname. . . . 169
2.56 DBDEAD. . . . 171
2.57 dberrhandle. . . . 172
2.58 dbexit. . . . 176
2.59 dbfcmd. . . . 177
Usage for dbfcmd. . . . 178
2.60 DBFIRSTROW. . . . 180
2.61 dbfree_xlate. . . . 181
2.62 dbfreebuf. . . . 183
2.63 dbfreequal. . . . 184
2.64 dbfreesort. . . . 185
2.65 dbgetchar. . . . 186
2.66 dbgetcharset. . . . 188
2.67 dbgetloginfo. . . . 189
2.68 dbgetlusername. . . . 191
2.69 dbgetmaxprocs. . . . 192
2.70 dbgetnatlang. . . . 193
2.71 dbgetoff. . . . 194
2.72 dbgetpacket. . . . 196
2.73 dbgetrow. . . . 197
2.74 DBGETTIME. . . . 199
2.75 dbgetuserdata. . . . 200
2.76 dbhasretstat. . . . 201
2.77 dbinit. . . . 203
2.78 DBIORDESC. . . . 203
2.79 DBIOWDESC 205
2.80 DBISAVAIL. . . . 206
2.81 dbisopt. . . . 207

2.82 DBLASTROW. 209
2.83 dbload_xlate. 210
2.84 dbloadsort. 212
2.85 dblogin. 213
2.86 dbloginfree. 215
2.87 dbmny4add. 216
2.88 dbmny4cmp. 217
2.89 dbmny4copy. 218
2.90 dbmny4divide. 220
2.91 dbmny4minus. 221
2.92 dbmny4mul. 222
2.93 dbmny4sub. 224
2.94 dbmny4zero. 225
2.95 dbmnyadd. 226
2.96 dbmnycmp. 228
2.97 dbmnycopy. 229
2.98 dbmnydec. 230
2.99 dbmnydivide. 232
2.100 dbmnydown. 233
2.101 dbmnyinc. 235
2.102 dbmnyinit. 236
2.103 dbmnymaxneg. 238
2.104 dbmnymaxpos. 239
2.105 dbmnyminus. 240
2.106 dbmnymul. 241
2.107 dbmnyndigit. 243
2.108 dbmnyscale. 247
2.109 dbmnysub. 249
2.110 dbmnyzero. 250
2.111 dbmonthname. 251
2.112 DBMORECMDS. 253
2.113 dbmoretext. 254
2.114 dbmsghandle. 255
2.115 dbname. 259
2.116 dbnextrow. 260
2.117 dbnpcreate. 262
2.118 dbnpdefine. 264
2.119 dbnullbind. 266
2.120 dbnumalts. 267
2.121 dbnumcols. 269
2.122 dbnumcompute. 270

2.123 DBNUMORDERS. 271
2.124 dbnumrets. 272
2.125 dbopen. 274
Usage for dbopen. 275
2.126 dbordercol. 278
2.127 dbpoll. 279
Usage for dbpoll. 281
2.128 dbprhead. 283
2.129 dbprrow. 284
2.130 dbprtype. 286
2.131 dbqual. 288
2.132 DBRBUF. 291
2.133 dbreadpage. 292
2.134 dbreadtext. 293
2.135 dbrecftos. 296
2.136 dbrecvpassthru. 297
2.137 dbregdrop. 299
2.138 dbregexec. 301
2.139 dbreghandle. 303
2.140 dbreginit. 306
2.141 dbreglist. 308
2.142 dbregnowatch. 309
2.143 dbregparam. 311
2.144 dbregwatch. 315
2.145 dbregwatchlist. 318
2.146 dbresults. 320
2.147 dbretdata. 322
2.148 dbretlen. 325
2.149 dbretname. 327
2.150 dbretstatus. 329
2.151 dbrettype. 331
2.152 DBROWS. 333
2.153 DBROWTYPE. 334
2.154 dbrpcinit. 335
2.155 dbrpcparam. 337
2.156 dbrpcsend. 339
2.157 dbrpwclr. 341
2.158 dbrpwset. 342
2.159 dbsafestr. 344
2.160 dbsechandle. 345
Usage for dbsechandle. 346

2.161 dbsendpassthru. 349
2.162 dbservcharset. 351
2.163 dbsetavail. 352
2.164 dbsetbusy. 353
2.165 dbsetconnect. 355
2.166 dbsetdefcharset. 357
2.167 dbsetdeflang. 358
2.168 dbsetidle. 359
2.169 dbsetifile. 361
2.170 dbsetinterrupt. 362
Usage for dbsetinterrupt. 363
2.171 DBSETLAPP. 365
2.172 DBSETLCHARSET. 367
2.173 DBSETLENCRYPT. 368
2.174 DBSETLHOST. 369
2.175 DBSETLMUTUALAUTH. 370
2.176 DBSETLNATLANG. 371
2.177 DBSETLNETWORKAUTH. 372
2.178 dbsetloginfo. 373
2.179 dbsetlogintime. 375
2.180 DBSETLPACKET. 377
2.181 DBSETLPWD. 378
2.182 DBSETLSERVERPRINCIPAL. 379
2.183 DBSETLUSER. 380
2.184 dbsetmaxprocs. 382
2.185 dbsetnull. 383
2.186 dbsetopt. 385
2.187 dbsetrow. 387
2.188 dbsettime. 389
2.189 dbsetuserdata. 390
2.190 dbsetversion. 392
2.191 dbspid. 394
2.192 dbspr1row. 395
2.193 dbspr1rowlen. 397
2.194 dbsprhead. 398
2.195 dbsprline. 400
2.196 dbsqlexec. 402
2.197 dbsqlok. 404
Usage for dbsqlok. 405
2.198 dbsqlsend. 409
2.199 dbstrbuild. 410

2.200 dbstrcmp. 412
2.201 dbstrcpy. 414
2.202 dbstrlen. 416
2.203 dbstrsort. 417
2.204 dbtabbrowse. 419
2.205 dbtabcount. 420
2.206 dbtabname. 422
2.207 dbtabsource. 423
2.208 DBTDS. 425
2.209 dbtextsize. 426
2.210 dbtsnewlen. 427
2.211 dbtsnewval. 428
2.212 dbtsput. 430
2.213 dbtxptr. 431
2.214 dbtxtimestamp. 433
2.215 dbtxtsnewval. 434
2.216 dbtxtsput. 435
2.217 dbuse. 437
2.218 dbvarylen. 438
2.219 dbversion. 439
2.220 dbwillconvert. 440
2.221 dbwritepage. 442
2.222 dbwritetext. 443
2.223 dbxlate. 448
2.224 Errors. 450
Usage for Errors. 450
2.225 Options. 466
2.226 Types. 470

3 Bulk Copy Routines. 474
3.1 Introduction to Bulk Copy. 474
Transferring Data into the Database. 475
Transferring Data out of the Database to a Flat File. 476
3.2 List of Bulk Copy Routines. 477
3.3 bcp_batch. 478
3.4 bcp_bind. 480
3.5 bcp_colfmt. 483
3.6 bcp_colfmt_ps. 487
3.7 bcp_collen. 491
3.8 bcp_colptr. 492
3.9 bcp_columns. 493
3.10 bcp_control. 494

3.11 bcp_done. . . . 497
3.12 bcp_exec. . . . 498
3.13 bcp_getl. . . . 500
3.14 bcp_init. . . . 501
3.15 bcp_moretext. . . . 503
3.16 bcp_options. . . . 506
3.17 bcp_readfmt. . . . 507
3.18 bcp_sendrow. . . . 508
3.19 BCP_SETL. . . . 510
3.20 bcp_setxlate. . . . 511
3.21 bcp_writefmt. . . . 512

4 Two-Phase Commit Service. . . . 515
4.1 Programming Distributed Transactions. . . . 515
4.2 Commit Service and the Application Program. . . . 516
4.3 The Probe Process. . . . 517
4.4 Two-Phase Commit Routines. . . . 517
4.5 Specifying the Commit Server. . . . 519
4.6 Two-Phase Commit Sample Program. . . . 519
4.7 Program Notes. . . . 524
Program Note 1. . . . 525
Program Note 2. . . . 525
Program Note 3. . . . 525
Program Note 4. . . . 525
Program Note 5. . . . 526
Program Note 6. . . . 526
Program Note 7. . . . 527
Program Note 8. . . . 527
4.8 abort_xact. . . . 527
4.9 build_xact_string. . . . 528
4.10 close_commit. . . . 530
4.11 commit_xact. . . . 531
4.12 open_commit. . . . 532
4.13 remove_xact. . . . 533
4.14 scan_xact. . . . 534
4.15 start_xact. . . . 535
4.16 stat_xact. . . . 537

5 Cursors. . . . 538
5.1 Cursor Overview. . . . 538
DB-Library Cursor Capability. . . . 539
Differences Between DB-Library Cursors and Browse Mode. . . . 539

Differences Between DB-Library and Client-Library Cursors. 539
5.2 Sensitivity to Change. 540
Static Cursor. 541
Keyset-Driven Cursor. 541
Dynamic Cursor. 541
Concurrency Control. 542
5.3 DB-Library Cursor Functions. 543
5.4 Holding Locks. 543
5.5 Stored Procedures Used by DB-Library Cursors. 544

6 DB-Library Error Messages. 545
6.1 20001. 552
6.2 20002. 553
6.3 20003. 554
6.4 20004. 555
6.5 20005. 556
6.6 20006. 557
6.7 20008. 558
6.8 20009. 559
6.9 20010. 560
6.10 20011. 561
6.11 20012. 562
6.12 20013. 563
6.13 20014. 564
6.14 20015. 565
6.15 20016. 566
6.16 20017. 567
6.17 20018. 568
6.18 20019. 569
6.19 20020. 570
6.20 20021. 571
6.21 20022. 572
6.22 20023. 573
6.23 20024. 574
6.24 20025. 575
6.25 20026. 576
6.26 20027. 577
6.27 20028. 578
6.28 20029. 579
6.29 20030. 580
6.30 20031. 581
6.31 20033. 582

6.32 20034. . . . 583
6.33 20035. . . . 584
6.34 20036. . . . 585
6.35 20037. . . . 586
6.36 20038. . . . 587
6.37 20039. . . . 588
6.38 20040. . . . 589
6.39 20041. . . . 590
6.40 20042. . . . 591
6.41 20043. . . . 592
6.42 20044. . . . 593
6.43 20045. . . . 594
6.44 20046. . . . 595
6.45 20047. . . . 596
6.46 20048. . . . 597
6.47 20049. . . . 598
6.48 20050. . . . 599
6.49 20051. . . . 600
6.50 20052. . . . 601
6.51 20053. . . . 602
6.52 20054. . . . 603
6.53 20055. . . . 604
6.54 20056. . . . 605
6.55 20060. . . . 606
6.56 20061. . . . 607
6.57 20062. . . . 608
6.58 20063. . . . 609
6.59 20064. . . . 610
6.60 20065. . . . 611
6.61 20066. . . . 612
6.62 20067. . . . 613
6.63 20068. . . . 614
6.64 20069. . . . 615
6.65 20070. . . . 616
6.66 20071. . . . 617
6.67 20072. . . . 618
6.68 20073. . . . 619
6.69 20074. . . . 620
6.70 20075. . . . 621
6.71 20076. . . . 622
6.72 20077. . . . 623

6.73 20078. . . . 624
6.74 20079. . . . 625
6.75 20080. . . . 626
6.76 20081. . . . 627
6.77 20082. . . . 628
6.78 20083. . . . 629
6.79 20084. . . . 630
6.80 20085. . . . 631
6.81 20086. . . . 632
6.82 20087. . . . 633
6.83 20088. . . . 634
6.84 20091. . . . 635
6.85 20092. . . . 636
6.86 20093. . . . 637
6.87 20094. . . . 638
6.88 20095. . . . 639
6.89 20096. . . . 640
6.90 20097. . . . 641
6.91 20098. . . . 642
6.92 20099. . . . 643
6.93 20100. . . . 644
6.94 20101. . . . 645
6.95 20102. . . . 646
6.96 20103. . . . 647
6.97 20104. . . . 648
6.98 20105. . . . 649
6.99 20106. . . . 650
6.100 20107. . . . 651
6.101 20108. . . . 652
6.102 20109. . . . 653
6.103 20110. . . . 654
6.104 20111. . . . 655
6.105 20112. . . . 656
6.106 20113. . . . 657
6.107 20114. . . . 658
6.108 20115. . . . 659
6.109 20116. . . . 660
6.110 20117. . . . 661
6.111 20118. . . . 662
6.112 20119. . . . 663
6.113 20120. . . . 664

6.114 20121. 665
6.115 20122. 666
6.116 20123. 667
6.117 20124. 668
6.118 20125. 669
6.119 20126. 670
6.120 20127. 671
6.121 20128. 672
6.122 20129. 673
6.123 20130. 674
6.124 20131. 675
6.125 20132. 676
6.126 20133. 677
6.127 20134. 678
6.128 20135. 679
6.129 20136. 680
6.130 20137. 681
6.131 20138. 682
6.132 20139. 683
6.133 20140. 684
6.134 20141. 685
6.135 20142. 686
6.136 20143. 687
6.137 20144. 688
6.138 20145. 689
6.139 20146. 690
6.140 20147. 691
6.141 20148. 692
6.142 20149. 693
6.143 20150. 694
6.144 20151. 695
6.145 20152. 696
6.146 20153. 697
6.147 20154. 698
6.148 20155. 699
6.149 20156. 700
6.150 20157. 701
6.151 20158. 702
6.152 20159. 703
6.153 20160. 704
6.154 20161. 705

6.155 20162. 706
6.156 20163. 707
6.157 20164. 708
6.158 20165. 709
6.159 20166. 710
6.160 20167. 711
6.161 20168. 712
6.162 20169. 713
6.163 20170. 714
6.164 20171. 715
6.165 20172. 716
6.166 20173. 717
6.167 20174. 718
6.168 20175. 719
6.169 20176. 720
6.170 20177. 721
6.171 20178. 722
6.172 20179. 723
6.173 20180. 724
6.174 20181. 725
6.175 20182. 726
6.176 20183. 727
6.177 20184. 728
6.178 20185. 729
6.179 20186. 730
6.180 20187. 731
6.181 20188. 732
6.182 20189. 733
6.183 20190. 734
6.184 20191. 735
6.185 20192. 736
6.186 20193. 737
6.187 20194. 738
6.188 20195. 739
6.189 20196. 740
6.190 20197. 741
6.191 20198. 742
6.192 20199. 743
6.193 20200. 744
6.194 20201. 745
6.195 20202. 746

6.196 20203. 747
6.197 20204. 748
6.198 20205. 749
6.199 20206. 750
6.200 20207. 751
6.201 20208. 752
6.202 20209. 753
6.203 20210. 754
6.204 20211. 755
6.205 20212. 756
6.206 20213. 757
6.207 20214. 758
6.208 20215. 759
6.209 20216. 760
6.210 20217. 761
6.211 20218. 762
6.212 20219. 763
6.213 20220. 764
6.214 20221. 765
6.215 20222. 766
6.216 20223. 767
6.217 20224. 768
6.218 20225. 769
6.219 20226. 770
6.220 20227. 771
6.221 20228. 772
6.222 20229. 773
6.223 20230. 774
6.224 20231. 775
6.225 20232. 776
6.226 20233. 777
6.227 20234. 778
6.228 20235. 779
6.229 20236. 780
6.230 20237. 781
6.231 20238. 782
6.232 20239. 783
6.233 20240. 784
6.234 20241. 785
6.235 20242. 786
6.236 20243. 787

6.237 20244. 788
6.238 20245. 789
6.239 20246. 790
6.240 20247. 791
6.241 20248. 792
6.242 20249. 793
6.243 20250. 794
6.244 20251. 795
6.245 20252. 796
6.246 20253. 797
6.247 20254. 798
6.248 20255. 799
6.249 20256. 800
6.250 20257. 801
6.251 20258. 802
6.252 20259. 803
6.253 20260. 804
6.254 20261. 805
6.255 20262. 806
6.256 20263. 807
6.257 20264. 808
6.258 20265. 809
6.259 20266. 810
6.260 20267. 811
6.261 20268. 812
6.262 20269. 813
6.263 20270. 814
6.264 20271. 815
6.265 20272. 816
6.266 20273. 817
6.267 20274. 818
6.268 20275. 819
6.269 20276. 820
6.270 20277. 821
6.271 20278. 822
6.272 20279. 823
6.273 20280. 824
6.274 20282. 825
6.275 20283. 826
6.276 20284. 827
6.277 20285. 828

6.278 20286. 829
6.279 20287. 830
6.280 20288. 831
6.281 20289. 832
6.282 20290. 833
6.283 20291. 834
6.284 20292. 835
6.285 20293. 836
6.286 20294. 837
6.287 20295. 838
6.288 20296. 839
6.289 20297. 840
6.290 20298. 841
6.291 20299. 842
6.292 20300. 843
6.293 20301. 844
6.294 20302. 845

1 DB-Library

Review the information about the DB-Library.

Related Information

Client/Server Architecture [page 18]
Types of Clients [page 19]
Types of Servers [page 19]
Open Client and SAP Open Server solution Products [page 19]
Data Structures for Communicating with Servers [page 22]
Writing a DB-Library/C Program [page 22]
DB-Library/C Routines [page 26]
MIT Kerberos on DB-Library [page 55]
Sample Programs [page 56]

1.1 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.

1.2 Types of Clients

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

Clients include:

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

1.3 Types of Servers

The SAP product line includes servers and tools for building servers.

Servers include:

- SAP Adaptive Server Enterprise (ASE) is a database server. SAP ASE manages information stored in one or more databases.
- SAP Open Server solution provides the tools and interfaces required to create a custom server, also called an "SAP Open Server application."

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

SAP ASE and SAP Open Server applications are similar in the following ways:

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

But they also differ:

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

1.4 Open Client and SAP Open Server solution Products

SAP provides two families of products to write client and server application programs.

Two families are:

- Open Client
- SAP Open Server

Related Information

Open Client [page 20]
SAP Open Server Solution [page 20]
Open Client Libraries [page 20]
What is in DB-Library/C? [page 21]
Comparing the Library approach to Embedded SQL [page 21]

1.4.1 Open Client

Open Client provides customer applications, third-party products, and other SAP products with the interfaces required to communicate with SAP ASE and SAP Open Server.

Open Client has two components: programming interfaces and network services.

The programming interfaces component of Open Client consists of libraries designed for use in writing client applications: Client-Library, DB-Library, and CS-Library.

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

1.4.2 SAP Open Server Solution

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

The programming interfaces component of SAP Open Server contains Server-Library and CS-Library.

> **i Note**
>
> Both Open Client and SAP Open Server include CS-Library, which contains utility routines that are useful to both client and server applications.

SAP Open Server network services are transparent.

1.4.3 Open Client Libraries

Review the information about the 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 SAP 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 include at least one call to CS-Library, because Client-Library routines use a structure, which is allocated in CS-Library.

1.4.4 What is in DB-Library/C?

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

i Note

DB-Library provides source code compatibility for older SAP applications. SAP encourages you to implement new applications with Client-Library or Embedded SQL.

DB-Library includes routines that send commands to SAP ASE and SAP Open Server applications and others that process the results of those commands. Other routines handle error conditions, perform data conversion, and provide various information about the interaction of the application 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.

1.4.5 Comparing the Library approach to Embedded SQL

Review the instructions to use Open Client library application or an Embedded SQL application to send SQL commands to SAP Adaptive Server Enterprise.

Context

You can either choose an Open Client library application or an Embedded SQL application to send SQL commands to SAP ASE.

Embedded SQL is a superset of Transact-SQL. An Embedded SQL application includes Embedded SQL commands intermixed with the host language statements of the application. 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.

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.

1.5 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.

1.6 Writing a DB-Library/C Program

You can write 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.

Procedure

1. Log on to a server.

2. Place the language commands into a buffer and send them to the server.
3. Process 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. Handle the DB-Library/C errors and server messages.
5. Close the connection with the server.

Results

The following example shows the basic framework of many DB-Library/C applications. The program opens a connection to an SAP 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
*/
dbsqlexec(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 following example illustrates the 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` appears first in the file. This file defines symbolic constants such as function return values, described in the reference pages in 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, which are meant to be used only by the DB-Library/C routines and cannot be directly accessed by the program. The definition of the DBPROCESS structure is defined in the `sybdb.h` file . As discussed earlier, the DBPROCESS structure is manipulated only through DB-Library/C routines; you cannot access its components directly. To ensure compatibility with future releases of DB-Library/C, use the contents of `sybdb.h` as documented in the reference pages in Routines.
  The third header file in the example, `syberror.h`, contains error severity values and is included if the program refers to those values.
- `dbinit` – This routine initializes DB-Library/C. It is 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, include this call at the start of all DB-Library/C programs.
- `dberrhandle` and `dbmsghandle` – `dberrhandle` installs a user-supplied error-handling routine, which is called whenever the application encounters a DB-Library/C error. Similarly, `dbmsghandle` installs a message-handling routine, which is 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 uses to log on 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 uses when logging on to DBSETLAPP. Sets the name of the application, which appears in the `sysprocesses` table of SAP ASE. 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 on to the server. It returns a DBPROCESS structure, which serves as the conduit for information between the application and the server. After this routine is called, the

application is connected with SAP ASE and can now send Transact-SQL commands to SAP ASE and process any results.

- `cmdb` – This routine fills the command buffer with Transact-SQL commands, which can then be sent to SAP ASE. Each succeeding call to `dbcmd` simply adds the supplied text to the end of any text already in the buffer. It is your 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.
- `dbsqlexec` – 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`) gets 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 the 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. With 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 takes 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 is 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.

Related Information

DB-Library/C Datatypes [page 26]

Routines [page 57]

Routines [page 57]

dbinit [page 203]

dberrhandle [page 172]

dbmsghandle [page 255]

dblogin [page 213]

DBSETLUSER [page 380]

DBSETLPWD [page 378]
DBSETLAPP [page 365]
dbopen [page 274]
dbcmd [page 110]
dbsqlexec [page 402]
dbresults [page 320]
dbbind [page 88]
dbnextrow [page 260]
dbexit [page 176]

1.6.1 DB-Library/C Datatypes

DB-Library/C defines datatypes for SAP 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 are compatible.

See Types for a list of SAP ASE datatypes and corresponding DB-Library/C program variable types. See the *SAP 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 SAP ASE 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.

Related Information

Types [page 470]
dbconvert_ps [page 128]

1.7 DB-Library/C Routines

Review the number of categories in 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 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 Bulk Copy Routines
- Two-Phase Commit Service, described in Two-Phase Commit Service

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

Related Information

Initialization [page 28]

Command Processing [page 31]

Results Processing [page 34]

Message and Error Handling [page 40]

Information Retrieval [page 41]

Browse Mode [page 44]

Text and Image Handling [page 46]

Datatype Conversion [page 48]

Process Control Flow [page 48]

Remote Procedure Call [page 49]

Registered Procedure Call [page 50]

Gateway Passthrough Routines [page 51]

Datetime and Money [page 52]

Cleanup [page 53]

Secure Support [page 54]

Miscellaneous Routines [page 54]

Two-Phase Commit Service Special Library [page 55]

Remote Procedure Call [page 49]

Registered Procedure Call [page 50]
Gateway Passthrough Routines [page 51]
Initialization [page 28]
Message and Error Handling [page 40]
Information Retrieval [page 41]
Datatype Conversion [page 48]
Process Control Flow [page 48]
Datetime and Money [page 52]
Cleanup [page 53]
Secure Support [page 54]
Miscellaneous Routines [page 54]
Routines [page 57]
Bulk Copy Routines [page 474]
Two-Phase Commit Service [page 515]
Command Processing [page 31]
Results Processing [page 34]
Browse Mode [page 44]
Text and Image Handling [page 46]

1.7.1 Initialization

The initialization routines set up and define the connection between the application program and a server.

The routines handle tasks, such 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`.

Related Information

Initializing DB-Library/C [page 29]
Setting up the LOGINREC [page 29]
Establishing a Server Connection [page 30]

1.7.1.1 Initializing DB-Library/C

Use the following top level routines set up the internal environment of the DB-Library.

Context

1. `dbinit` – Initialize the underlying structures used by DB-Library/C.
2. `dbsetversion` – Specify a DB-Library version level.
3. `dbsetmaxprocs` – Set the maximum number of simultaneously open DBPROCESS structures.
4. `dbgetmaxprocs` – Indicate the current maximum number of simultaneously open DBPROCESS structures.

For more information about these routine, see Routines [page 57]

Related Information

dbinit [page 203]
dbsetversion [page 392]
dbsetmaxprocs [page 382]
dbgetmaxprocs [page 192]

1.7.1.2 Setting up the LOGINREC

Use the following routines to set up LOGINREC.

Context

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

1. `dblogin` – allocate a LOGINREC structure for subsequent use by `dbopen`.
2. `DBSETLUSER` – set the server user name in the LOGINREC.
3. `DBSETLPWD` – set the server password in the LOGINREC.
4. `DBSETLAPP` – set the application name in the LOGINREC.
5. `DBSETLHOST` – set the host name in the LOGINREC.
6. `DBSETLCHARSET` – set the character set in the LOGINREC.
7. `DBSETLPACKET` – set the Tabular Data Stream (TDS) packet size for an application.
8. `dbgetpacket` – return the current TDS packet size.

9. `dbrpwset` – add a remote password to a LOGINREC structure. The server will use this password when it performs a remote procedure call on another server.
10. `dbrpwclr` – clear all remote passwords from a LOGINREC structure.
11. `dbloginfree` – free a LOGINREC structure.

For more information about these routine, see Routines [page 57]

Related Information

dblogin [page 213]
DBSETLUSER [page 380]
DBSETLPWD [page 378]
DBSETLAPP [page 365]
DBSETLHOST [page 369]
DBSETLCHARSET [page 367]
DBSETLPACKET [page 377]
dbgetpacket [page 196]
dbrpwset [page 342]
dbrpwclr [page 341]
dbloginfree [page 215]

1.7.1.3 Establishing a Server Connection

Use the following routines to set up and open a connection to a remote server.

Context

The application calls the following routines:

1. `dbsetifile` – specify the interfaces file that `dbopen` uses to connect to a server.
2. `dbsetlogintime` – set the number of seconds DB-Library/C waits for a server to respond to a request by `dbopen` for a DBPROCESS connection.
3. `dbopen` – set up communication with the network, log into a server using the LOGINREC, initialize any options specified in the LOGINREC, and allocate 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.
4. `dbuse` – set 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.

For more information about these routine, see Routines [page 57]

Related Information

dbsetifile [page 361]
dbsetlogintime [page 375]
dbopen [page 274]
dbuse [page 437]

1.7.2 Command Processing

Review the instructions about how an application can communicate with a server through language commands.

Context

For Adaptive Server Enterprise, the language is Transact-SQL.

For SAP Open Server, the language is whatever the SAP 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.

Related Information

Building the Command Batch [page 31]
Accessing the Command Batch [page 32]
Executing the Command Batch [page 33]
Setting and Clearing Command Options [page 33]

1.7.2.1 Building the Command Batch

Use the following routines to build the command batch.

Context

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

1. `dbcmd` – add text to the command buffer. You can call `dbcmd` repeatedly to add multiple commands, or parts of commands. The text added with each successive call is concatenated to the earlier text.
2. `dbfcmd` – add 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.
3. `dbfreebuf` – clear 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`.

For more information about these routine, see Routines [page 57]

Related Information

dbcmd [page 110]
dbfcmd [page 177]
dbfreebuf [page 183]

1.7.2.2 Accessing the Command Batch

Use the following routines to examine and copy parts of the command buffer.

Context

1. `dbgetchar` – returns a pointer to a particular character in the command buffer.
2. `dbstrlen` – returns the length of the command buffer.
3. `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.

Related Information

dbgetchar [page 186]
dbstrlen [page 416]
dbstrcpy [page 414]
Setting Results Timeouts [page 40]

1.7.2.3 Executing the Command Batch

Use the following routines to execute the command batch.

Context

After you enter the language commands buffer, you can send them to a server for execution.

1. `dbsqlsend` – sends the contents of the command buffer to a server for execution. Unlike `dbsqlexec`, this routine does not wait for a response from the server. When `dbsqlsend` returns SUCCEED, call `dbsqlok` to verify the correctness of the command batch.
2. `dbpoll` – when called between `dbsqlsend` (or `dbrpcsend`) and `dbsqlok`, checks if a server response has arrived for a DBPROCESS.
3. `dbsqlok` – 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 `dbsqlok` call, the application must call `dbresults` to process the results.
4. `dbsqlexec` – sends the contents of the command buffer to a server for execution. Once `dbsqlexec` has returned SUCCEED, call `dbresults` to process the results. Calling `dbsqlexec` is equivalent to calling `dbsqlsend` followed by `dbsqlok`.

Related Information

dbsqlsend [page 409]
dbpoll [page 279]
dbsqlok [page 404]
dbsqlexec [page 402]

1.7.2.4 Setting and Clearing Command Options

Use the following routines to set and clear command option.

Context

The application can set a number of SAP Adaptive Server Enterprise (ASE) and DB-Library/C command options. Among them are DBPARSEONLY, which causes SAP ASE 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 [page 466].

1. `dbsetopt` – sets an option

2. `dbclropt` – clears an option
3. `dbisopt` – determines whether a particular option is set

Related Information

Options [page 466]
dbsetopt [page 385]
dbclropt [page 108]
dbisopt [page 207]

1.7.3 Results Processing

Review the instructions on how the result processing takes place after command processing.

Context

Once a command batch has been executed in the server, indicated by `dbsqlexec` or `dbsqlok` 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 the `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 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.

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

Transact-SQL Command	Required DB-Library/C Functions
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.

Related Information

Setting up the Results [page 35]
Getting Result Data [page 36]
Reading Result Rows [page 37]
Canceling Results [page 38]
Handling Stored Procedure Results [page 39]
Setting Results Timeouts [page 40]
dbresults [page 320]

1.7.3.1 Setting up the Results

Review the instructions to set up the results.

Context

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

Related Information

dbresults [page 320]

1.7.3.2 Getting Result Data

Review the instructions to get result data.

Context

The simplest way to get result data is to bind result columns to program variables.

Use the `dbbind` and `dbaltbind` routines. When the application calls `dbnextrow` to read a result row (see *Reading Result Rows*), DB-Library/C places 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 with `dbdatlen` and `dbadlen`, which return the length of the data and are described in the section *Information Retrieval*. 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 `dbcoltype` and `dbaltop`.

For more information about these routine, see Routines [page 57]

Related Information

Information Retrieval [page 41]
dbbind [page 88]
dbbind_ps [page 93]
dbaltbind [page 68]
dbaltbind_ps [page 74]
dbdata [page 151]
dbadata [page 64]
dbnullbind [page 266]
dbanullbind [page 87]
dbsetnull [page 383]
dbprtype [page 286]
Reading Result Rows [page 37]

1.7.3.3 Reading Result Rows

Review the instructions to read result rows.

Context

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 is copied into the variables. Alternatively, the data is accessible through `dbdata` or `dbadata`.

Rows read in by `dbnextrow` are 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` overwrites 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 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`.

For more information about these routine, see Routines [page 57]

Related Information

dbnextrow [page 260]
DBCURROW [page 136]
dbprhead [page 283]
dbprrow [page 284]

1.7.3.4 Canceling Results

Review the instructions to cancel results.

Context

The following routines cancel results:

- `dbcancel` – cancels results from the current command batch. This routine cancels all the commands in the current batch.
- `dbcanquery` – 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`.

Related Information

dbcancel [page 101]
dbcanquery [page 102]

1.7.3.5 Handling Stored Procedure Results

Review how to handle stored procedure results in DB-Library/C.

Context

A call to a stored procedure is made through either a remote procedure call, discussed in *Remote Procedure Call Processing*, or a Transact-SQL `execute` command. The call can generate the following types of results:

1. A stored procedure that contains `select` statements returns result rows in the usual fashion. Each successive call to `dbresults` accesses the set of rows from the next `select` statement in the stored procedure. These rows can be processed, as usual, with `dbnextrow`.
2. 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.
3. 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.

Related Information

Remote Procedure Call [page 49]
dbnumrets [page 272]
dbretdata [page 322]

dbretlen [page 325]
dbrettype [page 331]
dbretname [page 327]
dbretstatus [page 329]
dbhasretstat [page 201]

1.7.3.6 Setting Results Timeouts

By default, DB-Library waits indefinitely for the results of a server command to arrive.

Context

Use the following routines to specify a finite timeout period:

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

Related Information

dbsettime [page 389]
DBGETTIME [page 199]

1.7.4 Message and Error Handling

Review the types of error messages that you may encounter when using DB-Library/C.

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

- Server messages and errors, 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 SAP Adaptive Server Enterprise (ASE) messages, use the Transact-SQL command:

  ```
  select * from sysmessages
  ```

  For a list of SAP ASE messages, see the *SAP Adaptive Server Enterprise System Administration Guide*. For a list of Open Server messages, see the *SAP 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 [page 452].

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:

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

We strongly recommend 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.
  However, even an application that uses centralized error and message handling needs 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, write a message handler and an error handler and install them using the `dbmsghandle` and `dberrhandle` routines.

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.

Related Information

Errors [page 450]
dbmsghandle [page 255]
dberrhandle [page 172]
DBDEAD [page 171]

1.7.5 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.

Related Information

Regular Result Column Information [page 42]

Compute Result Column Information [page 42]
Row Buffer Information [page 43]
Command State Information [page 43]

1.7.5.1 Regular Result Column Information

Review the information about regular result column.

These routines can be called after `dbsqlexec` 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 used 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.

Related Information

dbnumcols [page 269]
dbcolname [page 116]
dbcollen [page 115]
dbcoltype [page 119]
dbdatlen [page 168]
dbvarylen [page 438]

1.7.5.2 Compute Result Column Information

The routines can be called after `dbsqlexec` returns SUCCEED.

The routines are:

- `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 used with `dbadata`. The value returned by `dbadlen` is different for each compute row read by `dbnextrow`.

Related Information

DBROWTYPE [page 334]
dbnumcompute [page 270]
dbnumalts [page 267]
dbbylist [page 99]
dbaltop [page 82]
dbalttype [page 84]
dbaltlen [page 81]
dbaltcolid [page 79]
dbadlen [page 66]

1.7.5.3 Row Buffer Information

The following macros return information that can be useful when manipulating result rows in buffers.

Macros are:

- `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.

Related Information

DBFIRSTROW [page 180]
DBLASTROW [page 209]
dbgetrow [page 197]
dbclrbuf [page 107]

1.7.5.4 Command State Information

Review the routines that 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 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.

Related Information

DBCURCMD [page 135]
dbgetoff [page 194]
DBMORECMDS [page 253]
DBCMDROW [page 112]
DBROWS [page 333]
DBCOUNT [page 134]
DBNUMORDERS [page 271]
dbordercol [page 278]

1.7.6 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 is transferred from the database into program variables before it can be browsed and updated.

Context

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 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 a database table that can be browsed, 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.

Procedure

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.

Results

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*.

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.
- `dbcolbrowse` – indicates whether the source of a result column is updatable through browse mode.
- `dbcolsource` – 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.

Related Information

Sample Programs [page 56]
dbqual [page 288]
dbfreequal [page 184]
dbtsnewval [page 428]
dbtsnewlen [page 427]
dbtsput [page 430]
dbcolbrowse [page 113]
dbcolsource [page 118]
dbtabbrowse [page 419]
dbtabcount [page 420]
dbtabname [page 422]
dbtabsource [page 423]

1.7.7 Text and Image Handling

The `text` and `image` SAP Adaptive Server Enterprise (ASE) datatypes are designed to hold large text or image values.

Context

The `text` datatype holds up to 2,147,483,647 bytes of printable characters; the `image` datatype holds up to 2,147,483,647 bytes of binary data.

As they are large, `text` and `image` values are not 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."

Procedure

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 has 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 throws 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`.

Results

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 SAP ASE.
- `dbwritetext` – sends a `text` or an `image` value to SAP ASE.
- `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.

Related Information

dbreadtext [page 293]
dbwritetext [page 443]
dbmoretext [page 254]
dbtxptr [page 431]
dbtxtimestamp [page 433]
dbtxtsnewval [page 434]
dbtxtsput [page 435]

1.7.8 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 [page 470].

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 converts it 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.

Related Information

Types [page 470]
dbconvert_ps [page 128]
dbconvert [page 124]
dbwillconvert [page 440]

1.7.9 Process Control Flow

Review the routines that 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.

Related Information

dbsetbusy [page 353]
dbsetidle [page 359]
dbsetinterrupt [page 362]
DBIORDESC [page 203]
DBIOWDESC [page 205]
DBRBUF [page 291]

1.7.10 Remote Procedure Call

A remote procedure call is a call to a stored procedure residing on a remote server.

An application or another server calls the remote procedure. A remote procedure call made by an application has the same effect as a `execute` command: It executes the stored procedure, generating results accessible through `dbresults`. However, a remote procedure call is more efficient than a `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 a `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 `dbsqlok`, checks if a server response has arrived for a DBPROCESS.
- `dbsqlok` – waits for results from the server and verifies the correctness of the instructions the server is responding to. This routine is used with `dbsqlsend`, `dbrpcsend`, and `dbmoretext`. After a successful `dbsqlok` call, the application must call `dbresults` to process the results.

Related Information

dbrpcinit [page 335]
dbrpcparam [page 337]
dbrpcsend [page 339]
dbpoll [page 279]
dbsqlok [page 404]

1.7.11 Registered Procedure Call

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

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 SAP 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 name of the procedure and the arguments it was called with.

> **i 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 SAP 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 SAP 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, SAP 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. SAP 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 SAP Open Server
- Drop a registered procedure
- List all registered procedures defined in SAP 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 SAP 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 SAP 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.

Related Information

dbnpcreate [page 262]
dbnpdefine [page 264]
dbregdrop [page 299]
dbreglist [page 308]
dbreghandle [page 303]
dbreginit [page 306]
dbregnowatch [page 309]
dbregparam [page 311]
dbregexec [page 301]
dbregwatch [page 315]
dbregwatchlist [page 318]
dbpoll [page 279]

1.7.12 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` SAP Open Server Server-Library routines:

- `dbrecvpassthru` – receives a TDS packet from SAP Open Server.
- `dbsendpassthru` – sends a TDS packet to SAP Open Server.

See the *SAP Open Server Server-Library/C Reference Manual* for descriptions of `srvrecvpassthru` and `srvsendpassthru`.

Related Information

dbrecvpassthru [page 297]
dbsendpassthru [page 349]

1.7.13 Datetime and Money

Review the information about routines for 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.
- `dbmnyscale` – multiplies a DBMONEY value and adds a specified amount.

Related Information

dbdate4cmp [page 153]
dbdate4zero [page 154]
dbmny4add [page 216]
dbmny4cmp [page 217]
dbmny4copy [page 218]
dbmny4divide [page 220]
dbmny4minus [page 221]
dbmny4mul [page 222]
dbmny4sub [page 224]
dbmny4zero [page 225]
dbmnydec [page 230]
dbmnydown [page 233]
dbmnyinc [page 235]
dbmnyinit [page 236]
dbmnymaxneg [page 238]
dbmnymaxpos [page 239]
dbmnyndigit [page 243]
dbmnyscale [page 247]

1.7.14 Cleanup

Review the routines that connects between the application and a server.

Routines:

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

Related Information

dbexit [page 176]
dbclose [page 106]

1.7.15 Secure Support

Review the routines that provide security for DB-Library applications running against SAP Adaptive Server Enterprise (ASE).

Routines are:

- `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.

> i 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`.

Related Information

DBSETLENCRYPT [page 368]
dbsechandle [page 345]
bcp_options [page 506]

1.7.16 Miscellaneous Routines

Review the routines that are 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.

Related Information

dbsetavail [page 352]
DBISAVAIL [page 206]
dbname [page 259]
dbchange [page 104]
dbsetuserdata [page 390]
dbgetuserdata [page 200]
dbreadpage [page 292]
dbwritepage [page 442]
dbsetconnect [page 355]

1.7.17 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 *Two-Phase Commit Service*.

Related Information

Two-Phase Commit Service [page 515]

1.8 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.

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

Related Information

Installing MIT-Kerberos on DB-Library [page 56]

1.8.1 Installing MIT-Kerberos on DB-Library

The following steps provide basic information on installing MIT Kerberos on DB-Library.

Context

See *Installation and Release Bulletin* for Sybase SDK DB-Lib Kerberos Authentication Option 15.5.

Procedure

1. In DB-Library, include `sybdbn.h` instead of `sybdb.h`.
2. Using `dbsetversion`, set the DB-Library version to DBVERSION_100 or above.
3. Call one or more of the following APIs:

   -
     ```
     DBSETLNETWORKAUTH(LOGINREC *<loginrec>, DBBOOL <enable>)
     ```

   -
     ```
     DBSETLMUTUALAUTH(LOGINREC *<loginrec>, DBBOOL <enable>)
     ```

   -
     ```
     DBSETLSERVERPRINCIPAL(LOGINREC *<loginrec>, char *<name>)
     ```
4. Recompile DB-Library.

1.9 Sample Programs

Sample programs that demonstrates 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.

2 Routines

Review the information for each DB-Library routine.

Related Information

db12hour [page 62]
dbadata [page 64]
dbadlen [page 66]
dbaltbind [page 68]
dbaltbind_ps [page 74]
dbaltcolid [page 79]
dbaltlen [page 81]
dbaltop [page 82]
dbalttype [page 84]
dbaltutype [page 85]
dbanullbind [page 87]
dbbind [page 88]
dbbind_ps [page 93]
dbbufsize [page 98]
dbbylist [page 99]
dbcancel [page 101]
dbcanquery [page 102]
dbchange [page 104]
dbcharsetconv [page 105]
dbclose [page 106]
dbclrbuf [page 107]
dbclropt [page 108]
dbcmd [page 110]
DBCMDROW [page 112]
dbcolbrowse [page 113]
dbcollen [page 115]
dbcolname [page 116]
dbcolsource [page 118]
dbcoltype [page 119]
dbcoltypeinfo [page 121]
dbcolutype [page 123]
dbconvert [page 124]
dbconvert_ps [page 128]

DBCOUNT [page 134]
DBCURCMD [page 135]
DBCURROW [page 136]
dbcursor [page 137]
dbcursorbind [page 139]
dbcursorclose [page 142]
dbcursorcolinfo [page 143]
dbcursorfetch [page 144]
dbcursorinfo [page 147]
dbcursoropen [page 148]
dbdata [page 151]
dbdate4cmp [page 153]
dbdate4zero [page 154]
dbdatechar [page 155]
dbdatecmp [page 157]
dbdatecrack [page 158]
dbdatename [page 161]
dbdateorder [page 163]
dbdatepart [page 165]
dbdatezero [page 167]
dbdatlen [page 168]
dbdayname [page 169]
DBDEAD [page 171]
dberrhandle [page 172]
dbexit [page 176]
dbfcmd [page 177]
DBFIRSTROW [page 180]
dbfree_xlate [page 181]
dbfreebuf [page 183]
dbfreequal [page 184]
dbfreesort [page 185]
dbgetchar [page 186]
dbgetcharset [page 188]
dbgetloginfo [page 189]
dbgetlusername [page 191]
dbgetmaxprocs [page 192]
dbgetnatlang [page 193]
dbgetoff [page 194]
dbgetpacket [page 196]
dbgetrow [page 197]
DBGETTIME [page 199]
dbgetuserdata [page 200]
dbhasretstat [page 201]

dbinit [page 203]
DBIORDESC [page 203]
DBIOWDESC [page 205]
DBISAVAIL [page 206]
dbisopt [page 207]
DBLASTROW [page 209]
dbload_xlate [page 210]
dbloadsort [page 212]
dblogin [page 213]
dbloginfree [page 215]
dbmny4add [page 216]
dbmny4cmp [page 217]
dbmny4copy [page 218]
dbmny4divide [page 220]
dbmny4minus [page 221]
dbmny4mul [page 222]
dbmny4sub [page 224]
dbmny4zero [page 225]
dbmnyadd [page 226]
dbmnycmp [page 228]
dbmnycopy [page 229]
dbmnydec [page 230]
dbmnydivide [page 232]
dbmnydown [page 233]
dbmnyinc [page 235]
dbmnyinit [page 236]
dbmnymaxneg [page 238]
dbmnymaxpos [page 239]
dbmnyminus [page 240]
dbmnymul [page 241]
dbmnyndigit [page 243]
dbmnyscale [page 247]
dbmnysub [page 249]
dbmnyzero [page 250]
dbmonthname [page 251]
DBMORECMDS [page 253]
dbmoretext [page 254]
dbmsghandle [page 255]
dbname [page 259]
dbnextrow [page 260]
dbnpcreate [page 262]
dbnpdefine [page 264]
dbnullbind [page 266]

dbnumalts [page 267]
dbnumcols [page 269]
dbnumcompute [page 270]
DBNUMORDERS [page 271]
dbnumrets [page 272]
dbopen [page 274]
dbordercol [page 278]
dbpoll [page 279]
dbprhead [page 283]
dbprrow [page 284]
dbprtype [page 286]
dbqual [page 288]
DBRBUF [page 291]
dbreadpage [page 292]
dbreadtext [page 293]
dbrecftos [page 296]
dbrecvpassthru [page 297]
dbregdrop [page 299]
dbregexec [page 301]
dbreghandle [page 303]
dbreginit [page 306]
dbreglist [page 308]
dbregnowatch [page 309]
dbregparam [page 311]
dbregwatch [page 315]
dbregwatchlist [page 318]
dbresults [page 320]
dbretdata [page 322]
dbretlen [page 325]
dbretname [page 327]
dbretstatus [page 329]
dbrettype [page 331]
DBROWS [page 333]
DBROWTYPE [page 334]
dbrpcinit [page 335]
dbrpcparam [page 337]
dbrpcsend [page 339]
dbrpwclr [page 341]
dbrpwset [page 342]
dbsafestr [page 344]
dbsechandle [page 345]
dbsendpassthru [page 349]
dbservcharset [page 351]

dbsetavail [page 352]
dbsetbusy [page 353]
dbsetconnect [page 355]
dbsetdefcharset [page 357]
dbsetdeflang [page 358]
dbsetidle [page 359]
dbsetifile [page 361]
dbsetinterrupt [page 362]
DBSETLAPP [page 365]
DBSETLCHARSET [page 367]
DBSETLENCRYPT [page 368]
DBSETLHOST [page 369]
DBSETLMUTUALAUTH [page 370]
DBSETLNATLANG [page 371]
DBSETLNETWORKAUTH [page 372]
dbsetloginfo [page 373]
dbsetlogintime [page 375]
DBSETLPACKET [page 377]
DBSETLPWD [page 378]
DBSETLSERVERPRINCIPAL [page 379]
DBSETLUSER [page 380]
dbsetmaxprocs [page 382]
dbsetnull [page 383]
dbsetopt [page 385]
dbsetrow [page 387]
dbsettime [page 389]
dbsetuserdata [page 390]
dbsetversion [page 392]
dbspid [page 394]
dbspr1row [page 395]
dbspr1rowlen [page 397]
dbsprhead [page 398]
dbsprline [page 400]
dbsqlexec [page 402]
dbsqlok [page 404]
dbsqlsend [page 409]
dbstrbuild [page 410]
dbstrcmp [page 412]
dbstrcpy [page 414]
dbstrlen [page 416]
dbstrsort [page 417]
dbtabbrowse [page 419]
dbtabcount [page 420]

dbtabname [page 422]
dbtabsource [page 423]
DBTDS [page 425]
dbtextsize [page 426]
dbtsnewlen [page 427]
dbtsnewval [page 428]
dbtsput [page 430]
dbtxptr [page 431]
dbtxtimestamp [page 433]
dbtxtsnewval [page 434]
dbtxtsput [page 435]
dbuse [page 437]
dbvarylen [page 438]
dbversion [page 439]
dbwillconvert [page 440]
dbwritepage [page 442]
dbwritetext [page 443]
dbxlate [page 448]
Errors [page 450]
Options [page 466]
Types [page 470]

2.1 db12hour

Determine whether the specified language uses 12-hour or 24-hour time.

Syntax

```
DBBOOL db12hour(dbproc, language)

DBPROCESS *dbproc;
char *language;
```

Parameters

dbproc

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

language

The name of the language of interest.

Returns

"TRUE" if `<language>` uses 12-hour time, "FALSE" otherwise.

Usage

- `db12hour` returns "TRUE" if `<language>` uses 12-hour time, and "FALSE" if it uses 24-hour time.
- If `<language>` is NULL, `<dbproc>`'s current language is signified. If both `<language>` and `<dbproc>` are NULL, then DB-Library's default language (for any future calls to `dbopen`) is signified.
- `db12hour` is useful when retrieving and manipulating DBDATETIME values using `dbsqlexec`. 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.
- The following code fragment illustrates the use of `db12hour`:

```
 DBBOOL time_format;
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 */
dbsqlexec(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);
}
```

Related Information

dbdateorder [page 163]
dbdayname [page 169]
dbmonthname [page 251]
dbsetopt [page 385]

2.2 dbadata

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.

Returns

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

Related Information

dbadlen [page 66]
dbaltbind [page 68]
dbaltlen [page 81]
dbalttype [page 84]
dbgetrow [page 197]
dbnextrow [page 260]
dbnumalts [page 267]

2.3 dbadlen

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.

Returns

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 `dbaltlen` 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.
*/
dbsqlexec(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);
}
}
```

Related Information

dbadata [page 64]

dbaltlen [page 81]

dbalttype [page 84]

dbgetrow [page 197]

dbnextrow [page 260]

dbnumalts [page 267]

2.4 dbaltbind

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 or 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 receives the copy of the data from the DBPROCESS. The following table 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 is converted. For a list of the data conversions provided by DB-Library, see dbwillconvert [page 440].

> i 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.

Bind types (dbaltbind) lists the legal `<vartype>` values recognized by `dbaltbind`, along with the server and program variable types that each one refers to:

Bind Types (dbaltbind)

Vartype	Program Variable Type	Server Datatype
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

⚠ Caution

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` converts server data into the specified `<vartype>`.

The available representations for character data are shown in the following table. 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 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 contains an asterisk (“*”) to indicate the error.

Binary data may be stored in the following 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 contains 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 is 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 is copied.

Returns

SUCCEED or FAIL.

`dbaltbind` returns FAIL if:

- The column number is not valid.
- The data conversion specified by `<vartype>` is not legal.
- `<varaddr>` is NULL.

`<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 */
```

```
dbsqlexec(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 predefined 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.
    - 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 is set to indicate whether or not the column value was null.

Related Information

dbwillconvert [page 440]
dbaltbind_ps [page 74]
dbaltbind_ps [page 74]
Types [page 470]
dbsetnull [page 383]
dbnullbind [page 266]
dbadata [page 64]
dbaltbind_ps [page 74]
dbanullbind [page 87]
dbbind [page 88]
dbbind_ps [page 93]
dbconvert [page 124]
dbconvert_ps [page 128]
dbnullbind [page 266]
dbsetnull [page 383]
dbsetversion [page 392]
dbwillconvert [page 440]
Types [page 470]

2.5 dbaltbind_ps

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

Syntax

```
RETCODE dbaltbind_ps(dbproc, computeid, column,
                   vartype, varlen, varaddr,
                   typeinfo)

DBPROCESS     *dbproc;
int                        computeid;
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.

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 is converted. For a list of the data conversions provided by DB-Library, see thedbwillconvert [page 440] routine.

i 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 [page 470].

Bind types (dbaltbind_ps) lists the legal `<vartype>` values recognized by `dbaltbind_ps`, along with the server and program variable types that each one refers to:

Bind Types (dbaltbind_ps)

Vartype	Program Variable Type	Server Datatype
CHARBIND	DBCHAR	SYBCHAR
STRINGBIND	DBCHAR	SYBCHAR
NTBSTRINGBIND	DBCHAR	SYBCHAR
VARYCHARBIND	DBVARYCHAR	SYBCHAR
BINARYBIND	DBBINARY	SYBBINARY
VARYBINBIND	DBVARYBIN	SYBBINARY
TINYBIND	DBTINYINT	SYBINT1

Vartype	Program Variable Type	Server Datatype
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

⚠ Caution

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

Vartype	Program Type	Padding	Terminator
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 contains 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>.

Returns

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 dbsetnull [page 383] function for a list of the default substitution values.)

Related Information

dbwillconvert [page 440]
Types [page 470]
dbsetnull [page 383]
dbaltbind [page 68]
dbanullbind [page 87]
dbbind [page 88]
dbbind_ps [page 93]
dbconvert [page 124]
dbconvert_ps [page 128]
dbdata [page 151]
dbnullbind [page 266]
dbsetnull [page 383]
dbsetversion [page 392]
dbwillconvert [page 440]
Types [page 470]

2.6 dbaltcolid

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.

Returns

The select list ID for the compute column. The first column in a select list is 1. If either the `<computeid>` or the `<column>` value is invalid, `dbaltcolid` returns -1.

Usage

- This routine returns the select list ID for a compute column. For example, given the SQL statement:

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

  the call `dbaltcolid(<dbproc>`, 1, 1) will return 2, since “name” is the second column in the select list.

Related Information

dbadata [page 64]
dbadlen [page 66]
dbaltlen [page 81]
dbgetrow [page 197]
dbnextrow [page 260]
dbnumalts [page 267]
dbprtype [page 286]

2.7 dbaltlen

Return the maximum length of the data for a particular compute column.

Syntax

```
DBINT dbaltlen(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.

Returns

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.

Related Information

dbadata [page 64]
dbadlen [page 66]
dbalttype [page 84]
dbgetrow [page 197]
dbnextrow [page 260]
dbnumalts [page 267]

2.8 dbaltop

Return the type of aggregate operator for a particular compute column.

Syntax

```
int dbaltop(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.

Returns

A token value for the type of the compute column's aggregate operator. In case of error, `dbaltop` returns -1.

Usage

- This routine returns the type of aggregate operator for a particular column in a compute row. For example, given the SQL statement:

  ```
  select dept, name from employee
  ```

  ```
  order by dept, name
  ```

  ```
  compute count(name) by dept
  ```

  The call `dbaltop(<dbproc>, 1, 1)` returns 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 [page 286] reference page for a list of all token values and their equivalent token strings.

Related Information

dbprtype [page 286]
dbadata [page 64]
dbadlen [page 66]
dbaltlen [page 81]
dbnextrow [page 260]
dbnumalts [page 267]
dbprtype [page 286]

2.9 dbalttype

Return the datatype for a compute column.

Syntax

```
int dbalttype(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.

Returns

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.
- SYBDATETIMN 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.

`dbalttype` returns -1 if either the `<computeid>` or the `<column>` value is invalid.

Usage

- This routine returns the datatype for a compute column..
- `dbalttype` 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 [page 286] 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 `dbalttype(<dbproc>`, 1, 1) returns the token value SYBINT4, because counts are of SYBINT4 type. `dbprtype` converts SYBINT4 into the readable token string "int".

Related Information

Types [page 470]
dbprtype [page 286]
dbadata [page 64]
dbadlen [page 66]
dbaltlen [page 81]
dbnextrow [page 260]
dbnumalts [page 267]
dbprtype [page 286]
Types [page 470]

2.10 dbaltutype

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.

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.

Returns

The user-defined datatype of the specified compute column on success; a negative integer on error.

Usage

- `dbaltutype` returns the user-defined datatype for a compute column.
- For a description of how to add user-defined datatypes to the server databases or Server-Library programs, see the *SAP Adaptive Server Enterprise Reference Manual* or the *Open Server Server-Library/C Reference Manual*.
- `dbaltutype` is defined as type DBINT, since both the DB-Library datatype DBINT and user-defined datatypes are 32 bits long.

Related Information

dbalttype [page 84]
dbcolutype [page 123]

2.11 dbanullbind

Associate an indicator variable with a compute-row column.

Syntax

```
RETCODE dbanullbind(dbproc, computeid, column,
                    indicator)

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

Parameters

dbproc

A pointer to the DBPROCESS structure that provides the connection for a particular front-end or 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.

> **i Note**
>
> `<indicator>` is just the pointer to the indicator variable. It is the variable itself that is set.

Returns

SUCCEED or FAIL.

`dbanullbind` returns FAIL if either `<computeid>` or `<column>` is invalid.

Usage

- `dbanullbind` 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.

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

Related Information

dbadata [page 64]
dbadlen [page 66]
dbaltbind [page 68]
dbnextrow [page 260]
dbnullbind [page 266]

2.12 dbbind

Bind a regular result column to a program variable.

Syntax

```
RETCODE dbbind(dbproc, column, vartype, varlen,
             varaddr)

DBPROCESS     *dbproc;
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 or 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 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 is converted. For a list of the data conversions provided by DB-Library, seedbwillconvert [page 440].

i Note

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

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

Bind types (dbbind) lists the legal `<vartype>` values recognized by `dbbind`, along with the server and program variable types that each one refers to:

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

Vartype	Program Variable Type	Server Datatype
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

⚠ Caution

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>`.

i 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.

Returns

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 */
dbsqlexec(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 substitute when binding null values. The `dbsetnull` function allows you to explicitly set your own null substitution values.

Related Information

dbwillconvert [page 440]
dbbind_ps [page 93]
dbbind_ps [page 93]
Types [page 470]
dbsetnull [page 383]
dbaltbind [page 68]
dbaltbind_ps [page 74]
dbnullbind [page 266]
dbbind_ps [page 93]
dbconvert [page 124]
dbconvert_ps [page 128]
dbdata [page 151]
dbnullbind [page 266]
dbsetnull [page 383]
dbsetversion [page 392]
dbwillconvert [page 440]
Types [page 470]

2.13 dbbind_ps

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

Syntax

```
RETCODE dbbind_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 or 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 dbwillconvert [page 440].

For a list of the type definitions used by DB-Library, see Types [page 470].

Bind types (dbbind_ps) lists the legal `<vartype>` values recognized by `dbbind_ps`, along with the server and program variable types that each one refers to:

Bind Types (dbbind_ps)

Vartype	Program Variable type	Server Type
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

Vartype	Program Variable type	Server Type
BITBIND	DBBIT	SYBBIT
DATETIMEBIND	DBDATETIME	SYBDATETIME
SMALLDATETIMEBIND	DBDATETIME4	SYBDATETIME4
MONEYBIND	DBMONEY	SYBMONEY
SMALLMONEYBIND	DBMONEY4	SYBMONEY4
BOUNDARYBIND	DBCHAR	SYBBOUNDARY
SENSITIVITYBIND	DBCHAR	SYBSENSITIVITY

⚠ Caution

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>`.

i 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

Vartype	Program Type	Padding	Terminator
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 contains 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.

i 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>.

Returns

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 dbsetnull [page 383] function for a list of the default substitution values.

Related Information

dbwillconvert [page 440]
Types [page 470]
dbsetnull [page 383]
dbaltbind [page 68]
dbaltbind_ps [page 74]
dbanullbind [page 87]
dbbind [page 88]
dbconvert [page 124]
dbconvert_ps [page 128]
dbdata [page 151]
dbnullbind [page 266]
dbsetnull [page 383]
dbsetversion [page 392]
dbwillconvert [page 440]
Types [page 470]

2.14 dbbufsize

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 or server process. It contains all the information that DB-Library uses to manage communications and data between the front end and the server.

Returns

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.

Related Information

dbgetrow [page 197]
dbclrbuf [page 107]
dbgetrow [page 197]
dbsetopt [page 385]
Options [page 466]

2.15 dbbylist

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.

Returns

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

- `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.)
- `dbresults` must return SUCCEED before the application calls this routine.
- Assume the following command has been executed:

  ```
  select dept, name, year, sales from employee
  ```

  ```
  order by dept, name, year
  ```

  ```
  compute count(name) by dept,name
  ```

  The call `dbbylist` (`<dbproc>`, 1, &`<size>`) sets `<size>` 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.

Related Information

dbadata [page 64]
dbadlen [page 66]
dbaltlen [page 81]
dbalttype [page 84]
dbcolname [page 116]
dbgetrow [page 197]
dbnextrow [page 260]

2.16 dbcancel

Cancel the current command batch.

Syntax

```
RETCODE dbcancel(dbproc)
DBPROCESS     *dbproc;
```

Parameters

dbproc

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

Returns

SUCCEED or FAIL.

The most common reasons for failure are a dead DBPROCESS or a network error. `dbcancel` will also return FAIL if the server is dead.

Usage

- This routine cancels execution of the current command batch on the server and flushes any pending results. The application can call it after calling `dbsqlexec`, `dbsqlsend`, `dbsqlok`, `dbresults`, or `dbnextrow`.The `dbcancel` 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.
- `dbcancel` cancels all the commands in the current command batch. To cancel only the results from the current command, call `dbcanquery` instead.
- 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:
  - Set a time limit for server reads with `dbsettime`, and add a special case to your error handler function to respond to SYBETIME errors. See the reference pages for `dberrhandle` and `dbsettime` for details.
  - Use `dbsetinterrupt` to install custom interrupt handling. See dbsetinterrupt [page 362] for details.
- If you have set your own interrupt handler using `dbsetinterrupt`, you cannot call `dbcancel` in your interrupt handler. This causes the output from the server to DB-Library to become out of sync.

Related Information

dberrhandle [page 172]
dbsettime [page 389]
dbsetinterrupt [page 362]
dbsetinterrupt [page 362]
dbcanquery [page 102]
dbnextrow [page 260]
dbresults [page 320]
dbsetinterrupt [page 362]
dbsqlexec [page 402]
dbsqlok [page 404]
dbsqlsend [page 409]

2.17 dbcanquery

Cancel any rows pending from the most recently executed query.

Syntax

```
RETCODE dbcanquery(dbproc)

DBPROCESS    *dbproc;
```

Parameters

dbproc

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

Returns

SUCCEED or FAIL.

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

Usage

- This routine is an efficient way to throw away any unread rows that result from the most recently executed SQL query. Calling `dbcanquery` is equivalent to calling `dbnextrow` until it returns NO_MORE_ROWS, but `dbcanquery` is faster because it allocates no memory and executes no bindings to user data.
- If you have set your own interrupt handler using `dbsetinterrupt`, you cannot call `dbcanquery` 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 `dbnextrow`.
- `dbresults` must return SUCCEED before an application can call `dbcanquery`.
- To ignore all of the results from all of the commands in the current command batch, call `dbcancel` instead.

Related Information

dbcancel [page 101]
dbnextrow [page 260]
dbresults [page 320]
dbsetinterrupt [page 362]
dbsqlexec [page 402]

2.18 dbchange

Determine whether a command batch has changed the current database.

Syntax

```
char *dbchange(dbproc)

DBPROCESS    *dbproc;
```

Parameters

dbproc

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

Returns

A pointer to the null-terminated name of the new database, if any. If the database has not changed, NULL is returned.

Usage

- `dbchange` informs the program of a change in the current database. It does so by catching any instance of the Transact-SQL `use` command.
- 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 `dbsqlexec` 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`.

Related Information

dbname [page 259]
dbresults [page 320]
dbsqlexec [page 402]
dbsqlsend [page 409]
dbuse [page 437]

2.19 dbcharsetconv

Indicate whether the server is performing character set translation.

Syntax

```
DBBOOL dbcharsetconv(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/C uses to manage communications and data between the front end and the server.

Returns

"TRUE" if the server is performing character set translations; "FALSE" if it is not.

Usage

- If a client and a server are using the same character set, the server is not performing translation. In this case, `dbcharsetconv` returns "FALSE".
- To get the name of its own character set, a client can call `dbgetcharset`.

- To get the name of the server’s character set, a client can call `dbservcharset`.

Related Information

dbservcharset [page 351]
dbgetcharset [page 188]
DBSETLCHARSET [page 367]

2.20 dbclose

Close and deallocate a single DBPROCESS structure.

Syntax

```
void dbclose(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.

Returns

None.

Usage

- `dbclose` is the inverse of `dbopen`. It cleans up any activity associated with one DBPROCESS structure and deallocates the space. It also closes the corresponding network connection.

- To close every open DBPROCESS structure, use `dbexit` instead.
- `dbclose` does not deallocate space associated with a LOGINREC. To deallocate a LOGINREC, an application can call `dbloginfree`.
- Calling `dbclose` with an argument not returned by `dbopen` is sure to cause trouble.

Related Information

dbexit [page 176]
dbopen [page 274]

2.21 dbclrbuf

Drop rows from the row buffer.

Syntax

```
void dbclrbuf(dbproc, n)

DBPROCESS*    dbproc;
DBINT                   n;
```

Parameters

dbproc

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

<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.

Returns

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 dbgetrow [page 197] 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`.

Related Information

dbgetrow [page 197]
dbgetrow [page 197]
dbnextrow [page 260]
dbsetopt [page 385]
Options [page 466]

2.22 dbclropt

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 or 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 [page 466] 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* 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.

Returns

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*.

Related Information

Options [page 466]
Options [page 466]
Options [page 466]
dbisopt [page 207]
dbsetopt [page 385]
Options [page 466]

2.23 dbcmd

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.

Returns

SUCCEED or FAIL.

Related Information

Usage for dbcmd [page 111]
dbfcmd [page 177]
dbfreebuf [page 183]
dbgetchar [page 186]
dbstrcpy [page 414]
dbstrlen [page 416]
Options [page 466]

2.23.1 Usage for dbcmd

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 or cleared explicitly. 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 `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`.

2.24 DBCMDROW

Determine whether the current command can return rows.

Syntax

```
RETCODE DBCMDROW(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.

Returns

SUCCEED or FAIL, to indicate whether the command can return rows.

Usage

- `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.
- 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`.

Related Information

dbnextrow [page 260]
dbresults [page 320]
DBROWS [page 333]
DBROWTYPE [page 334]

2.25 dbcolbrowse

Determine whether the source of a regular result column is updatable through the DB-Library browse-mode facilities.

Syntax

```
DBBOOL dbcolbrowse(dbproc, colnum)

DBPROCESS    *dbproc;
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.

colnum

The number of the result column of interest. Column numbers start at 1.

Returns

"TRUE" or "FALSE."

Usage

- dbcolbrowse is one of the DB-Library browse mode routines. See Browse Mode [page 44] for a detailed discussion of browse mode.
- 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.
- 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:

  ```
  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.

Related Information

DB-Library [page 18]
dbcolsource [page 118]
dbqual [page 288]
dbtabbrowse [page 419]
dbtabcount [page 420]
dbtabname [page 422]
dbtabsource [page 423]
dbtsnewlen [page 427]
dbtsnewval [page 428]
dbtsput [page 430]

2.26 dbcollen

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.

Returns

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
*/
dbsqlexec(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);
}
```

Related Information

dbcolname [page 116]
dbcoltype [page 119]
dbdata [page 151]
dbdatlen [page 168]
dbnumcols [page 269]

2.27 dbcolname

Return the name of a regular result column.

Syntax

```
char *dbcolname(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.

Returns

A CHAR pointer to the null-terminated name of the particular column. If the column number is not in range, `dbcolname` returns NULL.

Usage

- This routine returns a pointer to the null-terminated name of a regular (that is, non-compute) result column.
- Here is a small program fragment that uses `dbcolname`:

```
 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
*/
dbsqlexec(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));
```

Related Information

dbcollen [page 115]

dbcoltype [page 119]

dbdata [page 151]

dbdatlen [page 168]

dbnumcols [page 269]

2.28 dbcolsource

Return a pointer to the name of the database column from which the specified regular result column was derived.

Syntax

```
char *dbcolsource(dbproc, colnum)

DBPROCESS   *dbproc;
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.

colnum

The number of the result column of interest. Column numbers start at 1.

Returns

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

- `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 *Browse Mode* for a detailed discussion of browse mode.
- `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:

  ```
  select author = au_lname from authors for browse
  ```

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:

```
dbcolsource(dbproc, 1)
```

This call returns a pointer to the string “au_lname”.

- `dbcolsource` is useful for ad hoc queries. If the query has been hard-coded into the program, this routine obviously is unnecessary.
- The application can call `dbcolsource` anytime after `dbresults`.
- The sample program `example7.c` contains a call to `dbcolsource`.

Related Information

DB-Library [page 18]
dbcolbrowse [page 113]
dbqual [page 288]
dbtabbrowse [page 419]
dbtabcount [page 420]
dbtabname [page 422]
dbtabsource [page 423]
dbtsnewlen [page 427]
dbtsnewval [page 428]
dbtsput [page 430]

2.29 dbcoltype

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.

Returns

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.
- SYBFLTN 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.
- `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.
*/
dbsqlexec(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));
}
```

Related Information

Types [page 470]
dbprtype [page 286]
dbcollen [page 115]
dbcolname [page 116]
dbdata [page 151]
dbdatlen [page 168]
dbnumcols [page 269]
dbprtype [page 286]
dbvarylen [page 438]
Types [page 470]

2.30 dbcoltypeinfo

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

Syntax

```
DBTYPEINFO * dbcoltypeinfo(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.

Returns

A pointer to a DBTYPEINFO structure that contains precision and scale values for a particular `numeric` or `decimal` column, or NULL if the specified column number is not in the result set.

A DBTYPEINFO structure is defined as follows:

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

If the datatype of the column is not `numeric` or `decimal`, the returned structure will contain meaningless values. Check that `dbcoltype` returns SYBNUMERIC or SYBDECIMAL before calling this function.

Usage

- This routine returns a pointer to a DBTYPEINFO structure that provides precision and scale information for a regular (that is, non-compute) result column of datatype `numeric` or `decimal`.
- The precision and scale values returned for columns with other datatypes will be meaningless. Check that `dbcoltype` returns SYBNUMERIC or SYBDECIMAL before calling `dbcoltypeinfo`.

Related Information

dbcollen [page 115]
dbcolname [page 116]
dbcoltype [page 119]
dbdata [page 151]
dbdatlen [page 168]
dbnumcols [page 269]
dbprtype [page 286]
dbvarylen [page 438]
Types [page 470]

2.31 dbcolutype

Return the user-defined datatype for a regular result column.

Syntax

```
DBINT dbcolutype(dbproc, column)

DBPROCESS   *dbproc;
int                        column;
```

Parameters

dbproc

A pointer to the DBPROCESS structure that provides the connection for a particular front-end or 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.

Returns

`<column>`'s user-defined datatype or a negative integer if `<column>` is not in range.

Usage

- `dbcolutype` 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 `sp_addtype` in the *SAP Adaptive Server Enterprise Reference Manual*.
- `dbcolutype` is defined as datatype DBINT to accommodate the size of user-defined datatypes. Both DBINT and user-defined datatypes are 32 bits long.
- The following code fragment illustrates the use of `dbcolutype`:

```
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.
*/
dbsqlexec(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));
}
```

Related Information

dbaltutype [page 85]

dbcoltype [page 119]

2.32 dbconvert

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 in the below table. .

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 *Type constants and program variable types*.

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.

Special values for `<destlen>`:

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

Returns

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 *Type constants and program variable types* 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.
- *Supported Datatype Conversions* 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 */
  dbsqlexec(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 [page 470] for a list of DB-Library datatypes and the corresponding Adaptive Server Enterprise datatypes. See the *SAP Adaptive Server Enterprise Reference Manual*.

Related Information

Types [page 470]
dbaltbind [page 68]
dbaltbind_ps [page 74]
dbbind [page 88]
dbbind_ps [page 93]
dbconvert_ps [page 128]
dberrhandle [page 172]
dbsetnull [page 383]

dbsetversion [page 392]
dbwillconvert [page 440]
Errors [page 450]
Types [page 470]

2.33 dbconvert_ps

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 *Supported Datatype Conversions*.

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 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 in *Supported Datatype Conversions*.

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.

Special values for destlen (dbconvert_ps) describes Special Values for <destlen>:

Special Values for destlen (dbconvert_ps)

Value of <destlen>	Applicable to	Meaning
-1	SYBCHAR, SYBBOUN-DARY, SBYSENSITIVITY	There is sufficient space available. The string will be trimmed of trailing blanks and given a terminating null.
-2	SYBCHAR	There is sufficient space available. 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>.

Returns

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 *Supported Datatype Conversion*).
  Type constants and program variable types shows type constants for server datatypes and the corresponding program variable types:

Server datatype constant	**Program variable type**
SYBCHAR	DBCHAR
SYBTEXT	DBCHAR
SYBBINARY	DBBINARY
SYBIMAGE	DBBINARY

Server datatype constant	Program variable type
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

⚠ Caution

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.

- *Supported Datatype Conversions* 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.)

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 for a list of DB-Library datatypes and the corresponding Adaptive Server Enterprise datatypes. See the *Adaptive Server Enterprise Reference Manual*.

Related Information

Types [page 470]
dbaltbind [page 68]
dbaltbind_ps [page 74]
dbbind [page 88]
dbbind_ps [page 93]
dbconvert [page 124]
dberrhandle [page 172]
dbsetnull [page 383]
dbsetversion [page 392]
dbwillconvert [page 440]
Errors [page 450]
Types [page 470]

2.34 DBCOUNT

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

Syntax

```
DBINT DBCOUNT(dbproc)

DBPROCESS       *dbproc;
```

Parameters

dbproc

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

Returns

The number of rows affected by the command, or -1. DBCOUNT will return -1 if any of the following are true:

- The Transact-SQL command fails for any reason, such as a syntax error.
- The command is one that never affects rows, such as a `print` command.
- The command executes a stored procedure that does not execute any `select` statements.
- The DBNOCOUNT option is `on`.

Usage

- Once the results of a command have been processed, you can call `DBCOUNT` 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 `dbnextrow` until it returned NO_MORE_ROWS, you can call this macro to find out how many rows were retrieved.
- If the current command is one that does not return rows, (for example, a `delete`), you can call DBCOUNT immediately after `dbresults`.
- If the command is one that executes a stored procedure, for example an `exec` or remote procedure call, DBCOUNT 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`.

Related Information

dbnextrow [page 260]
dbresults [page 320]
Options [page 466]

2.35 DBCURCMD

Return the number of the current command.

Syntax

```
int DBCURCMD(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.

Returns

The number of the current command.

Usage

- This macro returns the number of the command whose results are currently being processed.

- 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 `dbsqlexec` or `dbsqlsend`.

Related Information

DBCMDROW [page 112]
DBMORECMDS [page 253]
DBROWS [page 333]

2.36 DBCURROW

Return the number of the row currently being read.

Syntax

```
DBINT DBCURROW(dbproc)

DBPROCESS       *dbproc;
```

Parameters

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

Returns

The number of the current row. This routine returns 0 if no rows have been processed yet.

Usage

- 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.
- The row number is reset to 0 by each new call to `dbresults`.
- The row number grows by one every time `dbnextrow` returns REG_ROW or a `<computeid>`.
- 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. For more information, see dbgetrow [page 197] and dbsetrow [page 387].

Related Information

dbnextrow [page 260]
dbgetrow [page 197]
dbsetrow [page 387]
dbclrbuf [page 107]
DBFIRSTROW [page 180]
dbgetrow [page 197]
DBLASTROW [page 209]
dbnextrow [page 260]
dbsetopt [page 385]
Options [page 466]

2.37 dbcursor

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. Values for optype (dbcursor) lists the operation types.

Values for optype (dbcursor)

Symbolic value	Operation
CRS_UPDATE	Updates data.
CRS_DELETE	Deletes data.
CRS_INSERT	Inserts data.
CRS_REFRESH	Fetches another row in the buffer.
CRS_LOCKCC	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.

Returns

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.

For more information, see Cursors [page 538]

Related Information

Cursors [page 538]
dbcursorbind [page 139]
dbcursorclose [page 142]
dbcursorcolinfo [page 143]
dbcursorfetch [page 144]
dbcursorinfo [page 147]
dbcursoropen [page 148]

2.38 dbcursorbind

Register the binding information on the cursor columns.

Syntax

```
RETCODE dbcursorbind(hc, col, vartype, varlen,
                   poutlen, pvaraddr, typeinfo)

DBCURSOR    *hc;
int                         col;
int                        vartype;
DBINT                 varlen;
DBINT                *poutlen;
BYTE                 *pvaraddr;
DBTYPEINFO   *typeinfo;
```

Parameters

hc

Cursor handle created by dbcursoropen.

col

Number of the column to be bound to a program variable.

vartype

Binding type, which uses the same datatypes as the <vartype> parameter for dbbind and is bound by the same conversion rules. If this value is set to NOBIND for any column, the data is not bound. Instead, a pointer to the data is returned to the address in the corresponding <pvaraddr> entry for every row, and the length of the data is returned to the corresponding <varlen> array entry. This feature lets the application access the cursor data as it does with dbdata and dbdatalen.

varlen

Maximum length of variable-length datatype, such as CHARBIND, VARYCHARBIND, BINARYBIND, STRINGBIND, NTBSTRINGBIND, and VARYBINBIND. This parameter is ignored for fixed-length datatypes, such as INTBIND, FLT8BIND, MONEYBIND, BITBIND, SMALLBIND, and so on. It is also ignored if the vartype is NOBIND.

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>`.

Returns

SUCCEED or FAIL.

Usage

- If `dbcursorbind` is called more than once for any column, only the last call is effective.
- This function works almost the same as `dbbind` without cursors.

For more information, see Cursors [page 538]

Related Information

Cursors [page 538]
dbcursor [page 137]
dbcursorclose [page 142]

dbcursorcolinfo [page 143]
dbcursorfetch [page 144]
dbcursorinfo [page 147]
dbcursoropen [page 148]

2.39 dbcursorclose

Close the cursor associated with the given handle and release all the data belonging to it.

Syntax

```
void dbcursorclose(hc)

DBCURSOR     *hc;
```

Parameters

hc
Cursor handle created by `dbcursoropen`.

Returns

None.

Usage

- Closing a DBPROCESS connection with `dbcursorclose` automatically closes all the cursors associated with it. After issuing `dbcursorclose`, the cursor handle should not be used.

Related Information

Cursors [page 538]
dbcursor [page 137]

dbcursorbind [page 139]
dbcursorcolinfo [page 143]
dbcursorfetch [page 144]
dbcursorinfo [page 147]
dbcursoropen [page 148]

2.40 dbcursorcolinfo

Return column information for the specified column number in the open cursor.

Syntax

```
RETCODE dbcursorcolinfo(hcursor, column, colname,
                      coltype, collen, usertype)

DBCURSOR      *hcursor
DBINT               column;
DBCHAR           *colname;
DBINT               *coltype;
DBINT               *collen;
DBINT               *usertype;
```

Parameters

hcursor

Cursor handle created by `dbcursoropen`.

column

Column number for which information is to be returned.

colname

Location where the name of the column is returned. The user should allocate space large enough to accommodate the column name.

coltype

Location where the column's datatype is returned.

collen

Location where the column's maximum length is returned.

usertype

Location where the column's user-defined datatype is returned.

Returns

SUCCEED or FAIL.

Usage

- Any of the parameters `<colname>`, `<coltype>`, `<collen>`, or `<usertype>` can be set to NULL, in which case the information for that variable is not returned.

For more information, see Cursors [page 538]

Related Information

Cursors [page 538]
dbcursor [page 137]
dbcursorbind [page 139]
dbcursorclose [page 142]
dbcursorfetch [page 144]
dbcursorinfo [page 147]
dbcursoropen [page 148]

2.41 dbcursorfetch

Fetch a block of rows into the program variables declared by the user in `dbcursorbind`.

Syntax

```
RETCODE dbcursorfetch(hc, fetchtype, rownum)

DBCURSOR     *hc;
DBINT               fetchtype;
DBINT                rownum;
```

Parameters

hc

Cursor handle created by `dbcursoropen`.

fetchtype

Type of fetch chosen. The scroll option in `dbcursoropen` determines which of these values are legal. Values for fetchtype (dbcursorfetch) lists the various fetch types:

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 `<rownum>` 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 `<rownum>` 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 `<rownum>` 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.

Returns

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.

Usage

- 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.

For more information, see Cursors [page 538]

Related Information

dbcursoropen [page 148]
dbcursoropen [page 148]
Cursors [page 538]
dbcursor [page 137]
dbcursorbind [page 139]
dbcursorclose [page 142]
dbcursorcolinfo [page 143]
dbcursorinfo [page 147]
dbcursoropen [page 148]

2.42 dbcursorinfo

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

hcursor

Cursor handle created by `dbcursoropen`.

ncols

Location where the number of columns in the cursor is returned.

nrows

Location where the number of rows in the keyset is returned.

Returns

SUCCEED or FAIL.

Usage

- For fully keyset-driven cursors, the `<nrows>` parameter contains the number of rows in the keyset. For mixed or dynamic cursors, `<nrows>` 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.

For more information, see Cursors [page 538]

Related Information

Cursors [page 538]
dbcursor [page 137]
dbcursorbind [page 139]
dbcursorclose [page 142]
dbcursorcolinfo [page 143]
dbcursorfetch [page 144]
dbcursoropen [page 148]

2.43 dbcursoropen

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

```
DBCURSOR *dbcursoropen(dbproc, stmt, scrollopt,
                     concuropt, nrows, pstatus)

DBPROCESS    *dbproc;
BYTE                *stmt;
SHORT             scrollopt;
SHORT             concuropt;
USHORT          nrows;
DBINT               *pstatus
```

Parameters

dbproc

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

stmt

The `select` statement that defines a cursor.

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.

The possible values for `<scrollopt>`:

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 `<n>`	Keyset-driven cursor within (`<n>`*`<nrows>`) blocks, but fully dynamic outside the keyset.

concuropt

Definition of concurrency control. Values for concuropt (dbcursoropen) lists the possible values for `<concuropt>`:

Values for concuropt (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 `dbcursorfetch`.
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 the following table to inspect the status value:

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.

Returns

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.

For more information, see Cursors [page 538]

Related Information

Cursors [page 538]
dbcursor [page 137]
dbcursorbind [page 139]
dbcursorclose [page 142]
dbcursorcolinfo [page 143]
dbcursorfetch [page 144]
dbcursorinfo [page 147]
dbcursoropen [page 148]

2.44 dbdata

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.

Returns

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
*/
dbsqlexec(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.

Related Information

dbbind [page 88]
dbcollen [page 115]

dbcolname [page 116]
dbcoltype [page 119]
dbdatlen [page 168]
dbnumcols [page 269]

2.45 dbdate4cmp

Compare two DBDATETIME4 values.

Syntax

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

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

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.

d1

A pointer to a DBDATETIME4 value.

d2

A pointer to a DBDATETIME4 value.

Returns

If `<d1>` = `<d2>`, `dbdate4cmp` returns 0.

If `<d1>` < `<d2>`, `dbdate4cmp` returns -1.

If `<d1>` > `<d2>`, `dbdate4cmp` returns 1.

Usage

- `dbdate4cmp` compares two DBDATETIME4 values.
- The range of legal DBDATETIME4 values is from January 1, 1900 to June 6, 2079. DBDATETIME4 values have a precision of one minute.

Related Information

dbdatecmp [page 157]
dbmnycmp [page 228]
dbmny4cmp [page 217]

2.46 dbdate4zero

Initialize a DBDATETIME4 variable to Jan 1, 1900 12:00AM.

Syntax

```
RETCODE dbdate4zero(dbproc, dateptr)

DBPROCESS    *dbproc;
DBDATETIME4   *dateptr;
```

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.

dateptr

A pointer to the DBDATETIME4 variable to initialize.

Returns

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.

Related Information

dbdatezero [page 167]

2.47 dbdatechar

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. Date parts and their character representations (dbdatechar) 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.

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.

Returns

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`.

Related Information

dbconvert [page 124]
dbdata [page 151]
dbdatename [page 161]
dbdatecrack [page 158]

2.48 dbdatecmp

Compare two DBDATETIME values.

Syntax

```
int dbdatecmp(dbproc, d1, d2)

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

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.

d1

A pointer to a DBDATETIME value.

d2

A pointer to a DBDATETIME value.

Returns

If <d1> = <d2>, dbdatecmp returns 0.

If <d1> < <d2>, dbdatecmp returns -1.

If <d1> > <d2>, dbdatecmp returns 1.

Usage

- dbdatecmp compares two DBDATETIME values.
- 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).

Related Information

dbdate4cmp [page 153]
dbmnycmp [page 228]
dbmny4cmp [page 217]

2.49 dbdatecrack

Convert a machine-readable DBDATETIME value into user-accessible format.

Syntax

```
RETCODE dbdatecrack(dbproc, dateinfo, datetime)

DBPROCESS    *dbproc;
DBDATEREC    *dateinfo;
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.

dateinfo

A pointer to a DBDATEREC structure to contain the parts of `<datetime>`. 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 datesecond; /* 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 dbdatename or dbdayname plus dbmonthname.

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

datetime

A pointer to the DBDATETIME value of interest.

Returns

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");
dbsqlexec(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);
}
```

Related Information

dbdatename [page 161]

dbdayname [page 169]

dbmonthname [page 251]

dbconvert [page 124]

dbdata [page 151]

dbdatechar [page 155]

dbdatename [page 161]

dbdatepart [page 165]

2.50 dbdatename

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. Date parts and their character representations (dbdatename) 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.

Date parts and their character representations (dbdatename)

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

Date part	Symbol	Character representation of value
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.

Returns

The number of bytes placed into *`<charbuf>`.

In case of error, `dbdatename` returns -1.

Usage

- `dbdatename` 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 `datename` function.
- The following code fragment illustrates the use of `dbdatename`:

```
 dbcmd(dbproc, "select name, crdate from \
master..sysdatabases");
dbsqlexec(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 */
dbdatename(dbproc, datestring, DBDATE_YY,
(DBDATETIME *) (dbdata(dbproc, 2)));
printf("\tYear = %s.\n", datestring);
dbdatename(dbproc, datestring, DBDATE_QQ,
 (DBDATETIME *) (dbdata(dbproc, 2)));
printf("\tQuarter = %s.\n", datestring);
dbdatename(dbproc, datestring, DBDATE_MM,
 (DBDATETIME *) (dbdata(dbproc, 2)));
printf("\tMonth = %s.\n", datestring);
dbdatename(dbproc, datestring, DBDATE_DW,
 (DBDATETIME *) (dbdata(dbproc, 2)));
printf("\tDay of week = %s.\n", datestring);
dbdatename(dbproc, datestring, DBDATE_DD,
 (DBDATETIME *) (dbdata(dbproc, 2)));
printf("\tDay of month = %s.\n", datestring);
dbdatename(dbproc, datestring, DBDATE_DY,
 (DBDATETIME *) (dbdata(dbproc, 2)));
printf("\tDay of year = %s.\n", datestring);
dbdatename(dbproc, datestring, DBDATE_HH,
 (DBDATETIME *) (dbdata(dbproc, 2)));
printf("\tHour = %s.\n", datestring);
dbdatename(dbproc, datestring, DBDATE_MI,
 (DBDATETIME *) (dbdata(dbproc, 2)));
printf("\tMinute = %s.\n", datestring);
dbdatename(dbproc, datestring, DBDATE_SS,
 (DBDATETIME *) (dbdata(dbproc, 2)));
printf("\tSecond = %s.\n", datestring);
dbdatename(dbproc, datestring, DBDATE_MS,
 (DBDATETIME *) (dbdata(dbproc, 2)));
printf("\tMillisecond = %s.\n", datestring);
dbdatename(dbproc, datestring, DBDATE_WK,
 (DBDATETIME *) (dbdata(dbproc, 2)));
printf("\tWeek = %s.\n", datestring);
```

Related Information

dbconvert [page 124]
dbdata [page 151]
dbdatechar [page 155]
dbdatecrack [page 158]

2.51 dbdateorder

Return the date component order for a given language.

Syntax

```
char *dbdateorder(dbproc, language)

DBPROCESS     *dbproc;
char                    *language;
```

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.

language

The name of the language of interest.

Returns

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 `dbdateorder` string corresponds to their order in `<language>`'s default datetime format.

`dbdateorder` returns a NULL pointer on failure.

Usage

- `dbdateorder` returns a character string that describes the order in which the month, day, and year date components appear in the specified language. If `<language>` is NULL, the current language of the specified DBPROCESS is used. If both `<language>` and `<dbproc>` are NULL, DB-Library's default language is used.

  ⚠ Caution

  The date order string returned by `dbdateorder` is a pointer to DB-Library's internal data structures. Application programs should neither modify this string, nor free it.
- The following code fragment illustrates the use of `dbdateorder`:

```
/* Retrieve the date order from Adaptive Server Enterprise */

 printf("date-order: %s\n",

      (dbdateorder(DBPROCESS *)NULL, (char *)NULL));
```

Related Information

dbconvert [page 124]
dbdata [page 151]
dbdatechar [page 155]

dbdatecrack [page 158]

2.52 dbdatepart

Return the specified part of a DBDATETIME value as a numeric value.

Syntax

```
DBINT dbdatepart(dbproc, datepart, datetime)

DBPROCESS     *dbproc;
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.

datepart

The date component of interest. Date parts and their character representations (dbdatepart) 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.

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)

Date part	Symbol	Character representation of value
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.

Returns

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.

Related Information

dbconvert [page 124]
dbdata [page 151]
dbdatechar [page 155]
dbdatecrack [page 158]
dbdatename [page 161]

2.53 dbdatezero

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

Syntax

```
RETCODE dbdatezero(dbproc, dateptr)

DBPROCESS      *dbproc;
DBDATETIME      *dateptr;
```

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.

dateptr

A pointer to the DBDATETIME variable to initialize.

Returns

SUCCEED or FAIL.

`dbdatezero` returns FAIL if `<dateptr>` is NULL.

Usage

- `dbdatezero` initializes a DBDATETIME value to Jan 1, 1900 12:00:00:000AM.
- 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).

Related Information

dbdate4zero [page 154]

2.54 dbdatlen

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

Syntax

```
DBINT dbdatlen(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.

Returns

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.

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
*/
dbsqlexec(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);
}
```

Related Information

dbcollen [page 115]

dbcolname [page 116]

dbcoltype [page 119]

dbdata [page 151]

dbnumcols [page 269]

2.55 dbdayname

Determine the name of a specified weekday in a specified language.

Syntax

```
char *dbdayname(dbproc, language, daynum)

DBPROCESS    *dbproc;
char                   *language;
int                       daynum;
```

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.

language

The name of the desired language.

daynum

The number of the desired day. Day numbers range from 1 (Monday) to 7 (Sunday).

Returns

The name of the specified day on success; a NULL pointer on error.

Usage

- `dbdayname` returns the name of the specified day in the specified language. If `<language>` is NULL, `<dbproc>`'s current language is used. If both `<language>` and `<dbproc>` are NULL, then U.S. English is used.
- The following code fragment illustrates the use of `dbdayname`:

```
/*
** Retrieve the name of each day of the week in
** U.S. English.
*/
for (daynum = 1; daynum <= 7; daynum++)
printf("Day %d: %s\n", daynum,
dbdayname((DBPROCESS *)NULL, (char *)NULL,
daynum));
```

Related Information

db12hour [page 62]

dbdateorder [page 163]

dbmonthname [page 251]

DBSETLNATLANG [page 371]

2.56 DBDEAD

Determine whether a particular DBPROCESS is dead.

Syntax

```
DBBOOL DBDEAD(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.

Returns

"TRUE" or "FALSE."

Usage

- This macro indicates whether or not the specified DBPROCESS has been marked dead. It is particularly useful in user-supplied error handlers.
- 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.

  > **i Note**
  >
  > If there is no user-supplied error handler, a dead DBPROCESS will cause the affected DB-Library routines not to fail, but to abort.

- 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.

Related Information

dberrhandle [page 172]
Errors [page 450]

2.57 dberrhandle

Install a user function to handle DB-Library errors.

Syntax

```
int (*dberrhandle(handler))()

int           (*handler)();
```

Parameters

handler

A pointer to the user function that is called whenever DB-Library determines that an error has occurred. DB-Library calls this function with six parameters shown in Error handler parameters.

Error handler parameters

Parameter	Meaning
`<dbproc>`	The affected DBPROCESS. If there is no DBPROCESS associated with this error, this parameter is NULL.
`<severity>`	The severity of the error (datatype `<int>`). Error severities are defined in `syberror.h`.
`<dberr>`	The identifying number of the error (datatype `<int>`). Error numbers are defined in `sybdb.h`.
`<oserr>`	The operating-system-specific error number that describes the cause of the error (datatype `int`). If there is no relevant operating system error, the value of this parameter is DBNOERR.
`<dberrstr>`	A printable description of `<dberr>` (datatype `char *`).

Parameter	Meaning
`<oserrstr>`	A printable description of `<oserr>` (datatype `char *`).

The error handler must return one of the four values listed in Error handler returns, directing DB-Library to perform particular actions:

Error handler returns

Return	Action
INT_EXIT	Print an error message and abort the program. DB-Library also returns an error indication to the operating system. (Note to UNIX programmers: DB-Library does not leave a core file.
INT_CANCEL	Return FAIL from the DB-Library routine that caused the error. Returning INT_CANCEL on timeout errors kill the `<dbproc>`.
INT_TIMEOUT	Cancel the operation that caused the error but leave the `<dbproc>` in working condition. This return value is meaningful only for timeout errors (SYBETIME). In any other case, this value is considered an error, and is 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 is considered an error, and is treated as an INT_EXIT.

If the error handler returns any value besides these four, the program aborts.

Error handlers on the Windows platform must be declared with CS_PUBLIC, as shown in the following example. For portability, callback handlers on other platforms are 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);
}
}
```

Returns

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 calls this error handler immediately. 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 determines 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 exhibits its default error-handling behavior: It aborts 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 returns 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* 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.

Common errors provides information on when DB-Library/C calls an application's message and error handlers:

Common errors

Error or message	Message handler called?	Error handler called?
SQL syntax error.	Yes.	Yes (SYBESMSG). (Code your handler to ignore the message.)
SQL `print` statement.	Yes.	No.
SQL `raiserror`.	Yes.	No.
Server dies.	No.	Yes (SYBESEOF). (Code your handler to exit the application.)
Timeout from the server. **Note** The default timeout period is infinite. The error handler does not receive timeout notifications unless a timeout period is specified with dbsettime.	No.	Yes (SYBETIME). (To wait for another timeout period, code your handler to return -INT_CONTINUE.)
Deadlock on query.	Yes. (Code your handler to test for deadlock. See the dbsetuserdata for an example.)	Yes (SYBESMSG). (Code your handler to ignore the message.)
Timeout on login.	No.	Yes (SYBEFCON, SYBECONN).
Login fails (`dbopen`).	Yes.	Yes (SYBEPWD). (Code your handler to exit the application.)
Use database message.	Yes. (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, .)

Error or message	Message handler called?	Error handler called?
Fatal Server error (severity greater than 16).	Yes. (Code your handler to exit the application.)	Yes (SYBESMSG).

Related Information

Errors [page 450]
dbsettime [page 389]
dbsetuserdata [page 390]
DBDEAD [page 171]
dbmsghandle [page 255]
Errors [page 450]

2.58 dbexit

Close and deallocate all DBPROCESS structures, and clean up any structures initialized by `dbinit`.

Syntax

```
void dbexit()
```

Returns

None.

Usage

- `dbexit` calls `dbclose` repeatedly for all allocated DBPROCESS structures. `dbclose` cleans up any activity associated with a single DBPROCESS structure and deallocates the space.
- You can use `dbclose` directly to close just a single DBPROCESS structure.
- `dbexit` also cleans up any structures initialized by `dbinit`, releasing the memory associated with those structures. It must be the last DB-Library call in any application that calls `dbinit`.

- To ensure future compatibility and portability, SAP strongly recommends that all applications call `dbinit` and `dbexit`, no matter what their environment.
  For environments requiring `dbinit`, the application must not make any other DB-Library call after calling `dbexit`.

Related Information

dbclose [page 106]
dbinit [page 203]
dbopen [page 274]

2.59 dbfcmd

Add text to the DBPROCESS command buffer using C runtime library `sprintf`-type formatting.

Syntax

```
RETCODE dbfcmd(dbproc, cmdstring, args...)

DBPROCESS     *dbproc;
char                    *cmdstring;
???                     args...;
```

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 format string of the form used by the `sprintf` routine.

There is an optional and variable number of arguments to `dbfcmd`. The number and type of arguments required depends on the format specifiers included in the `<cmdstring>` argument. The arguments are passed directly to the C-library `sprintf` function. Neither `dbfcmd` nor the C compiler can type check these arguments. As with using `sprintf`, the programmer must ensure that each argument type matches the corresponding format specifier.

Returns

SUCCEED or FAIL.

Related Information

Usage for dbfcmd [page 178]

dbcmd [page 110]

dbfreebuf [page 183]

dbgetchar [page 186]

dbstrcpy [page 414]

dbstrlen [page 416]

Options [page 466]

2.59.1 Usage for dbfcmd

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.
- The following table lists the conversions supported by `dbfcmd`:

  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`.

i 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. 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 `dbfcmd`. To format commands that require more than eight `<args>`, call `dbfcmd` repeatedly. On some platforms, `dbfcmd` may allow more than eight `<args>` per call. For portable code, do not pass more than eight arguments.

Because it makes text substitutions, `dbfcmd` uses a working buffer in addition to the DBPROCESS command buffer. `dbfcmd` 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, `dbfcmd` allocates a working buffer 1200 bytes long. If the length of `<cmdstring>` is 34 bytes, `dbfcmd` allocates a working buffer 1024 bytes long. To work around this limitation:

```
sprintf (buffer, “%s”, SQL commmand”);
```

```
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 `dbfcmd`.

Note that the working buffer is not the same as the DBPROCESS command buffer. The working buffer is a temporary buffer used only by `dbfcmd` 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.

2.60 DBFIRSTROW

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 or server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

Returns

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

- This macro returns the number of the first row in the row buffer.
- 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.
- 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. For more information, seedbclrbuf [page 107].

Related Information

dbclrbuf [page 107]
dbclrbuf [page 107]
DBCURROW [page 136]
dbgetrow [page 197]
DBLASTROW [page 209]
dbnextrow [page 260]
dbsetopt [page 385]
Options [page 466]

2.61 dbfree_xlate

Free a pair of character set translation tables.

Syntax

```
RETCODE *dbfree_xlate(dbproc, xlt_tosrv, xlt_todisp)

DBPROCESS    *dbproc;
DBXLATE          *xlt_tosrv;
DBXLATE           *xlt_todisp;
```

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.

xlt_tosrv

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`.

xlt_todisp

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`.

Returns

SUCCEED or FAIL.

Usage

- This routine frees a pair of character set translation tables allocated by `dbload_xlate`.
- Character set translation tables translate characters between the server's standard character set and the display device's character set.
- The following code fragment illustrates the use of `dbfree_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: \
\n\t%s\n\n", destbuf);
dbfree_xlate((DBPROCESS *)NULL, xlt_tosrv,
xlt_todisp);
```

Related Information

dbload_xlate [page 210]

dbxlate [page 448]

2.62 dbfreebuf

Clear the command buffer.

Syntax

```
void dbfreebuf(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.

Returns

None.

Usage

- This routine clears a DBPROCESS command buffer by freeing any space allocated to it. It then sets the command buffer to NULL. Commands are added to the command buffer with the `dbcmd` or `dbfcmd` routine.
- After a call to `dbsqlexec` 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`.

Related Information

dbcmd [page 110]

dbfcmd [page 177]
dbgetchar [page 186]
dbsqlexec [page 402]
dbsqlsend [page 409]
dbstrcpy [page 414]
dbstrlen [page 416]
Options [page 466]

2.63 dbfreequal

Free the memory allocated by `dbqual`.

Syntax

```
void dbfreequal(qualptr)

char         *qualptr;
```

Parameters

qualptr

A pointer to the memory allocated by `dbqual`.

Returns

None.

Usage

- `dbfreequal` is one of the DB-Library browse mode routines. See Browse Mode [page 44] 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.

Related Information

DB-Library [page 18]
dbqual [page 288]

2.64 dbfreesort

Free a sort order structure allocated by `dbloadsort`.

Syntax

```
RETCODE dbfreesort(dbproc, sortorder)

DBPROCESS          *dbproc;
DBSORTORDER     *sortorder;
```

Parameters

dbproc

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

sortorder

A pointer to a DBSORTORDER structure allocated through `dbloadsort`.

Returns

SUCCEED or FAIL.

Usage

- `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.

- 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.

  > ⚠ Caution
  >
  > Application programs must not attempt to use operating-system facilities to free the *`<sortorder>` 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.

- The following code fragment illustrates the use of `dbfreesort`:

```
 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);
```

Related Information

dbloadsort [page 212]
dbstrcmp [page 412]
dbstrsort [page 417]

2.65 dbgetchar

Return a pointer to a character in the command buffer.

Syntax

```
char *dbgetchar(dbproc, n)

DBPROCESS     *dbproc;
int                         n;
```

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.

n

The position of the desired character in the command buffer. The first character position is 0.

Returns

`dbgetchar` returns a pointer to the `<n>`th character in the command buffer. If `<n>` is not in range, `dbgetchar` returns NULL.

Usage

- 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 `<n>`. The first character has position 0.
- 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.
- 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.

Related Information

dbcmd [page 110]
dbfcmd [page 177]
dbfreebuf [page 183]
dbstrcpy [page 414]
dbstrlen [page 416]

2.66 dbgetcharset

Get the name of the client character set from the DBPROCESS structure.

Syntax

```
char *dbgetcharset(dbproc)

DBPROCESS      *dbproc;
```

Parameters

dbproc

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

Returns

A pointer to the null-terminated name of the client character set, or NULL in case of error.

Usage

- `dbgetcharset` returns the name of the client's character set.
- 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 performs all conversions to and from its own character set when communicating with the client.
- An application can inform the server what character set it is using through DBSETLCHARSET.
- To determine if the server is performing character set translations, an application can call `dbcharsetconv`.
- To get the name of the server character set, an application can call `dbservcharset`.

Related Information

dbcharsetconv [page 105]

dblogin [page 213]
dbopen [page 274]
dbservcharset [page 351]
DBSETLCHARSET [page 367]

2.67 dbgetloginfo

Transfer Tabular Data Stream (TDS) login response information from a DBPROCESS structure to a newly allocated DBLOGINFO structure.

Syntax

```
RETCODE dbgetloginfo(dbproc, loginfo)

DBPROCESS     *dbproc;
DBLOGINFO      **loginfo;
```

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.

loginfo

The address of a DBLOGINFO pointer variable. `dbgetloginfo` sets the DBLOGINFO pointer to the address of a newly allocated DBLOGINFO structure.

Returns

SUCCEED or FAIL.

Usage

- `dbgetloginfo` transfers TDS login response information from a DBPROCESS structure to a newly allocated DBLOGINFO structure.

- An application must call `dbgetloginfo` only if 1) it is an SAP 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 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;
{
DBLOGINFO *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);
}
```

Related Information

dbrecvpassthru [page 297]
dbsendpassthru [page 349]
dbsetloginfo [page 373]

2.68 dbgetlusername

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

login

A pointer to a LOGINREC structure, which can be passed as an argument to `dbopen`. You can get a LOGINREC structure by calling `dblogin`.

name_buffer

A pointer to a buffer. The user name is copied from the LOGINREC structure to this buffer.

buffer_len

The length, in bytes, of the destination buffer.

Returns

The number of bytes copied into the destination buffer, not including the null-terminator.

If the user name is more than `<buffer_len>` -1 bytes long, `dbgetlusername` copies `<buffer_len>` -1 bytes into the destination buffer and returns DBTRUNCATED.

`dbgetlusername` returns FAIL if `<login>` is NULL, `<name_buffer>` is NULL, or `<buffer_len>` is less than 0.

Usage

- `dbgetlusername` copies the user name from LOGINREC structure into the `<name_buffer>` buffer.
- To set the user name in a LOGINREC structure, use DBSETLUSER.
- `dbgetlusername` copies a maximum of `<buffer_len>` -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.
- If the user name is in the LOGINREC is longer than `<buffer_len>` -1 bytes, `dbgetlusername` truncates the name and returns DBTRUNCATED.

Related Information

dblogin [page 213]
DBSETLUSER [page 380]

2.69 dbgetmaxprocs

Determine the current maximum number of simultaneously open DBPROCESSes.

Syntax

```
int dbgetmaxprocs()
```

Parameters

None.

Returns

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 `dbsetmaxprocs`.

Related Information

dbopen [page 274]
dbsetmaxprocs [page 382]

2.70 dbgetnatlang

Get the national language from the DBPROCESS structure.

Syntax

```
char* dbgetnatlang(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/C uses to manage communications and data between the front end and server.

Returns

A pointer to a character string representing the national language that the client DBPROCESS is using.

Usage

- `dbgetnatlang` returns a pointer to the name of the national language that a client is using.
- 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 DBSETLNATLANG.

Related Information

dblogin [page 213]
dbopen [page 274]
DBSETLNATLANG [page 371]

2.71 dbgetoff

Check for the existence of Transact-SQL constructs in the command buffer.

Syntax

```
int dbgetoff(dbproc, offtype, startfrom)

DBPROCESS       *dbproc;
DBUSMALLINT     offtype;
int                          startfrom;
```

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.

offtype

The type of offset you want to find. The types, which are defined in the header file `sybdb.h`, are:

OFF_SELECT OFF_FROM OFF_ORDER OFF_COMPUTE OFF_TABLE OFF_PROCEDURE OFF_STATEMENT OFF_PARAM OFF_EXEC

See *Options* for details.

startfrom

The point in the buffer at which to start looking. The command buffer begins at 0.

Returns

The character offset into the command buffer for the specified offset. If the offset is not found, -1 is returned.

Usage

- If the DBOFFSET option has been set, 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 `select` appears:

  ```
  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 */
  dbsqlexec(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.

Related Information

Options [page 466]

Options [page 466]

dbcmd [page 110]

dbgetchar [page 186]
dbsetopt [page 385]
dbstrcpy [page 414]
dbstrlen [page 416]
Options [page 466]
Options [page 466]

2.72 dbgetpacket

Return the TDS packet size currently in use.

Syntax

```
int dbgetpacket(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/C uses to manage communications and data between the front end and the server.

Returns

The TDS packet size currently in use.

Usage

- `dbgetpacket` returns the TDS packet size currently in use.
- 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.
- 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 SAP 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.

Related Information

DBSETLPACKET [page 377]

2.73 dbgetrow

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.

Returns

`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>`.
- 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.

Usage

- `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 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 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`.

Related Information

dbaltbind [page 68]
dbaltbind [page 68]
dbbind [page 88]
dbclrbuf [page 107]
DBCURROW [page 136]
DBFIRSTROW [page 180]
DBLASTROW [page 209]

dbnextrow [page 260]
dbsetrow [page 387]
Options [page 466]

2.74 DBGETTIME

Return the number of seconds that DB-Library will wait for a server response to a SQL command.

Syntax

```
int DBGETTIME()
```

Returns

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

- This routine returns the length of time in seconds that DB-Library waits for a server response during calls to `dbsqlexec`, `dbsqlok`, `dbresults`, and `dbnextrow`. The default timeout value is 0, which represents an infinite timeout period.
- The program can call `dbsettime` to change the timeout value.

Related Information

dbsettime [page 389]

2.75 dbgetuserdata

Return a pointer to user-allocated data from a DBPROCESS structure.

Syntax

```
BYTE *dbgetuserdata(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.

Returns

A generic BYTE pointer to the user's private data space. This pointer must have been previously saved with the `dbsetuserdata` routine.

Usage

- This routine returns, from a DBPROCESS structure, a pointer to user-allocated data. The application must have previously saved this pointer with the `dbsetuserdata` routine.
- `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.

Related Information

dbsetuserdata [page 390]

dbsetuserdata [page 390]

2.76 dbhasretstat

Determine whether the current command or remote procedure call generated a return status number.

Syntax

```
DBBOOL dbhasretstat(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.

Returns

“TRUE” or “FALSE”.

Usage

- 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 a `execute` command can generate a status number.
- The `dbretstatus` routine actually gets the status number. Stored procedures that complete, return a status number of 0. For a list of return status numbers, see the *SAP Adaptive Server Enterprise Reference Manual*.
- 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 dbsqlexec 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 is retrieved inside the dbresults loop, after all rows have been fetched with dbnextrow. The following code 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 */
```

Related Information

dbnextrow [page 260]

dbresults [page 320]

dbretdata [page 322]

dbretstatus [page 329]

dbrpcinit [page 335]

dbrpcparam [page 337]

dbrpcsend [page 339]

2.77 dbinit

Initialize DB-Library.

Syntax

```
RETCODE dbinit()
```

Returns

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, SAP strongly recommends that all applications call `dbinit`, no matter what their operating environment.

Related Information

dbexit [page 176]

2.78 DBIORDESC

(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

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.

Returns

An integer file descriptor used by the specified DBPROCESS to read data coming from the server.

Usage

- 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 (made using a call to `dbsqlsend`) and the server's response (read by calling `dbsqlok`, `dbresults`, or `dbnextrow`) may be significant. You may use this time to service other parts of your application. The DBIORDESC routine provides a way to obtain the I/O descriptor that a DBPROCESS uses to read the data stream from the server. This information may then be used with various operating system facilities (such as the UNIX `select` call) to allow the application to respond effectively to multiple input streams.
- `dbpoll` checks if a server response has arrived for any of an application's server connections (represented by DBPROCESS pointers). `dbpoll` is simpler to use than DBIORDESC. For this reason, and because DBIORDESC is non-portable, it is preferable to use `dbpoll`.
- The file descriptor returned by DBIORDESC may only be used with operating system facilities that do not 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 becomes hopelessly scrambled.
- An application can use the DB-Library DBRBUF routine, in addition to the UNIX `select` function, to help determine whether any more data from the server is available for reading.
- A companion routine, DBIOWDESC, provides access to the file descriptor used to write data to the server.

Related Information

dbcmd [page 110]
DBIOWDESC [page 205]
dbnextrow [page 260]

dbpoll [page 279]
DBRBUF [page 291]
dbresults [page 320]
dbsqlok [page 404]
dbsqlsend [page 409]

2.79 DBIOWDESC

(UNIX only) Provide program access to the UNIX file descriptor used by a DBPROCESS to write data to the server.

Syntax

```
int DBIOWDESC(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.

Returns

An integer file descriptor used by the specified DBPROCESS to write data to the server.

Usage

- 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 (made using a call to `dbsqlsend`) 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 `select` function) to allow the application to effectively utilize multiple input and output streams.

- The file descriptor returned by this routine may only be used with operating system facilities that do not 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 becomes scrambled.
- A companion routine, DBIORDESC, provides access to the file descriptor used to read data coming from the server. For some applications, another routine, `dbpoll` may be preferable to DBIORDESC.

Related Information

dbcmd [page 110]
DBIORDESC [page 203]
dbnextrow [page 260]
dbpoll [page 279]
dbresults [page 320]
dbsqlok [page 404]
dbsqlsend [page 409]

2.80 DBISAVAIL

Determine whether a DBPROCESS is available for general use.

Syntax

```
DBBOOL DBISAVAIL(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.

Returns

"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 set the DBPROCESS to "not available," but `dbsetavail` resets it to "available." This facility is useful when several parts of a program are attempting to share a single DBPROCESS.

Related Information

dbsetavail [page 352]

2.81 dbisopt

Check the status of a server or DB-Library option.

Syntax

```
DBBOOL dbisopt(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. Unlike in the functions `dbsetopt` and `dbclropt`, `<dbproc>` cannot be NULL here.

option

The option to be checked. See *Options* for the 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 [page 466] lists the 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.

Returns

"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 `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.

Related Information

Options [page 466]
Options [page 466]
Options [page 466]
dbclropt [page 108]
dbsetopt [page 385]
Options [page 466]

2.82 DBLASTROW

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.

Returns

The number of the last row in the row buffer. This routine returns 0 if there is an error.

Usage

- 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 not from the top of the row buffer.
- 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 returns the number of the row that is the last row in the row buffer.

Related Information

dbclrbuf [page 107]
DBCURROW [page 136]
DBFIRSTROW [page 180]
dbgetrow [page 197]

dbnextrow [page 260]
dbsetopt [page 385]
Options [page 466]

2.83 dbload_xlate

Load a pair of character set translation tables.

Syntax

```
RETCODE dbload_xlate(dbproc, srv_charset, xlate_name,
                   xlt_tosrv, xlt_todisp)

DBPROCESS     *dbproc;
char                    *srv_charset;
char                    *xlt_name;
DBXLATE             **xlt_tosrv;
DBXLATE             **xlt_todisp;
```

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.

srv_charset

A pointer to the name of the server's character set. `dbload_xlate` looks for a directory of this name in the `charsets` directory under the main Sybase installation directory. For example, if the server is using the iso_1 character set, `dbload_xlate` looks for `$SYBASE/charsets/iso_1`.

xlt_name

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.

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`.

Returns

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: \
\n\t%s\n\n", destbuf);
dbfree_xlate((DBPROCESS *)NULL, xlt_tosrv,
xlt_todisp);
```

Related Information

dbfree_xlate [page 181]
dbxlate [page 448]

2.84 dbloadsort

Load a server sort order.

Syntax

```
DBSORTORDER *dbloadsort(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.

Returns

A pointer to a DBSORTORDER structure on success, NULL on error.

Usage

- `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.
- `dbloadsort` allocates a DBSORTORDER structure to contain the server character set sort order information. The structure is freed using `dbfreesort`.
- The following code fragment illustrates the use of `dbloadsort`:

```
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);
```

Related Information

dbfreesort [page 185]
dbstrcmp [page 412]
dbstrsort [page 417]

2.85 dblogin

Allocates a login record for use in `dbopen`.

Syntax

```
LOGINREC *dblogin()
```

Returns

A pointer to a LOGINREC structure. `dblogin` returns NULL if the structure could not be allocated.

Usage

- This routine allocates a LOGINREC structure for use with `dbopen`.
- 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.

Related Information

dbloginfree [page 215]
dbopen [page 274]
dbrpwclr [page 341]
dbrpwset [page 342]
DBSETLAPP [page 365]
DBSETLCHARSET [page 367]
DBSETLHOST [page 369]
DBSETLNATLANG [page 371]
DBSETLPACKET [page 377]
DBSETLPWD [page 378]
DBSETLUSER [page 380]

2.86 dbloginfree

Free a login record.

Syntax

```
void dbloginfree(loginptr)
LOGINREC        *loginptr;
```

Parameters

loginptr

A pointer to a LOGINREC structure.

Returns

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.

Related Information

dblogin [page 213]
dbopen [page 274]

2.87 dbmny4add

Add two DBMONEY4 values.

Syntax

```
RETCODE dbmny4add(dbproc, m1, m2, sum)

DBPROCESS    *dbproc;
DBMONEY4      *m1;
DBMONEY4      *m2;
DBMONEY4      *sum;
```

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.

m1

A pointer to a DBMONEY4 value.

m2

A pointer to a DBMONEY4 value.

sum

A pointer to a DBMONEY4 variable to hold the result of the addition.

Returns

SUCCEED or FAIL.

`dbmny4add` returns FAIL in case of overflow, or if `<m1>`, `<m2>`, or `<sum>` is NULL.

Usage

- `dbmny4add` adds the `<m1>` and `<m2>` DBMONEY4 values and places the result in *`<sum>`.
- In case of overflow, `dbmny4add` returns FAIL and sets *`<sum>` to $0.00.
- 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.

Related Information

dbmny4sub [page 224]
dbmny4mul [page 222]
dbmny4divide [page 220]
dbmny4minus [page 221]
dbmny4add [page 216]
dbmny4sub [page 224]
dbmnymul [page 241]
dbmnydivide [page 232]
dbmnyminus [page 240]

2.88 dbmny4cmp

Compare two DBMONEY4 values.

Syntax

```
int dbmny4cmp(dbproc, m1, m2)

DBPROCESS     *dbproc;
DBMONEY4       *m1;
DBMONEY4       *m2;
```

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.

m1

A pointer to a DBMONEY4 value.

m2

A pointer to a DBMONEY4 value.

Returns

If `<m1>` = `<m2>`, `dbmny4cmp` returns 0.

If `<m1>` < `<m2>`, `dbmny4cmp` returns -1.

If `<m1>` > `<m2>`, `dbmny4cmp` returns 1.

Usage

- `dbmny4cmp` compares two DBMONEY4 values.
- 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.

Related Information

dbmnycmp [page 228]

2.89 dbmny4copy

Copy a DBMONEY4 value.

Syntax

```
RETCODE dbmny4copy(dbproc, src, dest)

DBPROCESS      *dbproc;
DBMONEY4        *src;
DBMONEY4          *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.

src

A pointer to the source DBMONEY4 value.

dest

A pointer to the destination DBMONEY4 variable.

Returns

SUCCEED or FAIL.

`dbmny4copy` returns FAIL if either `<src>` or `<dest>` is NULL.

Usage

- `dbmny4copy` copies the `<src>` DBMONEY4 value to the `<dest>` DBMONEY4 variable.
- 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.

Related Information

dbmnycopy [page 229]
dbmnyminus [page 240]
dbmny4minus [page 221]

2.90 dbmny4divide

Divide one DBMONEY4 value by another.

Syntax

```
RETCODE dbmny4divide(dbproc, m1, m2, quotient)

DBPROCESS      *dbproc;
DBMONEY4        *m1;
DBMONEY4        *m2;
DBMONEY4        *quotient;
```

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.

m1

A pointer to the DBMONEY4 value serving as dividend.

m2

A pointer to the DBMONEY4 value serving as divisor.

quotient

A pointer to a DBMONEY4 variable to hold the result of the division.

Returns

SUCCEED or FAIL.

`dbmny4divide` returns FAIL in case of overflow or division by zero, or if `<m1>`, `<m2>`, or `<quotient>` is NULL.

Usage

- dbmny4divide divides the <m1> DBMONEY4 value by the <m2> DBMONEY4 value and places the result in *<quotient>.
- In case of overflow or division by zero, dbmny4divide returns FAIL and sets *<quotient> 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.

Related Information

dbmny4add [page 216]
dbmny4sub [page 224]
dbmny4mul [page 222]
dbmny4minus [page 221]
dbmnyadd [page 226]
dbmnysub [page 249]
dbmnymul [page 241]
dbmnydivide [page 232]
dbmnyminus [page 240]

2.91 dbmny4minus

Negate a DBMONEY4 value.

Syntax

```
RETCODE dbmny4minus(dbproc, src, dest)

DBPROCESS    *dbproc;
DBMONEY4      *src;
DBMONEY4      *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.

src

A pointer to a DBMONEY4 value.

dest

A pointer to a DBMONEY4 variable to hold the result of the negation.

Returns

SUCCEED or FAIL.

`dbmny4minus` returns FAIL in case of overflow, or if `<src>` or `<dest>` is NULL.

Usage

- `dbmny4minus` negates the `<src>` DBMONEY4 value and places the result into *`<dest>`.
- In case of overflow, `dbmny4minus` returns FAIL. *`<dest>` is undefined in this case. An attempt to negate the maximum negative DBMONEY4 value will result in overflow.
- 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.

Related Information

dbmnyminus [page 240]
dbmnycopy [page 229]
dbmny4copy [page 218]

2.92 dbmny4mul

Multiply two DBMONEY4 values.

Syntax

```
RETCODE dbmny4mul(dbproc, m1, m2, product)
```

```
DBPROCESS     *dbproc;
DBMONEY4       *m1;
DBMONEY4       *m2;
DBMONEY4       *product;
```

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.

m1

A pointer to a DBMONEY4 value.

m2

A pointer to a DBMONEY4 value.

product

A pointer to a DBMONEY4 variable to hold the result of the multiplication.

Returns

SUCCEED or FAIL.

`dbmny4mul` returns FAIL in case of overflow, or if `<m1>`, `<m2>`, or `<product>` is NULL.

Usage

- `dbmny4mul` multiplies the `<m1>` DBMONEY4 value by the `<m2>` DBMONEY4 value and places the result in *`<product>`.
- In case of overflow, `dbmny4mul` returns FAIL and sets *`<product>` 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.

Related Information

dbmny4add [page 216]

dbmny4sub [page 224]
dbmny4divide [page 220]
dbmny4minus [page 221]
dbmnyadd [page 226]
dbmnysub [page 249]
dbmnymul [page 241]
dbmnydivide [page 232]
dbmnyminus [page 240]

2.93 dbmny4sub

Subtract one DBMONEY4 value from another.

Syntax

```
RETCODE dbmny4sub(dbproc, m1, m2, difference)

DBPROCESS    *dbproc;
DBMONEY4      *m1;
DBMONEY4      *m2;
DBMONEY4      *difference;
```

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.

m1

A pointer to the DBMONEY4 value to be subtracted from.

m2

A pointer to the DBMONEY4 value to subtract.

difference

A pointer to a DBMONEY4 variable to hold the result of the subtraction.

Returns

SUCCEED or FAIL.

`dbmny4sub` returns FAIL in case of overflow, or if `<m1>`, `<m2>`, or `<difference>` is NULL.

Usage

- `dbmny4sub` subtracts the `<m2>` DBMONEY4 value from the `<m1>` DBMONEY4 value and places the result in *`<difference>`.
- In case of overflow, `dbmny4sub` returns FAIL and sets *`<difference>` 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.

Related Information

dbmny4sub [page 224]
dbmny4mul [page 222]
dbmny4divide [page 220]
dbmny4minus [page 221]
dbmny4add [page 216]
dbmnysub [page 249]
dbmnymul [page 241]
dbmnydivide [page 232]
dbmnyminus [page 240]

2.94 dbmny4zero

Initialize a DBMONEY4 variable to $0.0000.

Syntax

```
RETCODE dbmny4zero(dbproc, mny4ptr)

DBPROCESS     *dbproc;
DBMONEY4       *mny4ptr;
```

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.

mny4ptr

A pointer to the DBMONEY4 value to initialize.

Returns

SUCCEED or FAIL.

`dbmny4zero` returns FAIL if `<mny4ptr>` is NULL.

Usage

- `dbmny4zero` initializes a DBMONEY4 value 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.

Related Information

dbmnyzero [page 250]

2.95 dbmnyadd

Add two DBMONEY values.

Syntax

```
RETCODE dbmnyadd(dbproc, m1, m2, sum)
```

```
DBPROCESS    *dbproc;
DBMONEY        *m1;
DBMONEY        *m2;
DBMONEY       *sum;
```

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.

m1

A pointer to a DBMONEY value.

m2

A pointer to a DBMONEY value.

sum

A pointer to a DBMONEY variable to hold the result of the addition.

Returns

SUCCEED or FAIL.

Usage

- `dbmnyadd` adds the `<m1>` and `<m2>` DBMONEY values and places the result in *`<sum>`.
- In case of overflow, `dbmnyadd` returns FAIL and sets *`<sum>` to $0.0000.
- 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.
- `dbmnyadd` returns FAIL in case of overflow, or if `<m1>`, `<m2>`, or `<sum>` is NULL.

Related Information

dbmnysub [page 249]
dbmnymul [page 241]

dbmnydivide [page 232]
dbmnyminus [page 240]
dbmny4add [page 216]
dbmny4sub [page 224]
dbmny4mul [page 222]
dbmny4divide [page 220]
dbmny4minus [page 221]

2.96 dbmnycmp

Compare two DBMONEY values.

Syntax

```
int dbmnycmp(dbproc, m1, m2)

DBPROCESS    *dbproc;
DBMONEY        *m1;
DBMONEY        *m2;
```

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.

m1

A pointer to a DBMONEY value.

m2

A pointer to a DBMONEY value.

Returns

If `<m1>` = `<m2>` `dbmnycmp` returns 0.

If `<m1>` < `<m2>` `dbmnycmp` returns -1.

If `<m1>` > `<m2>` `dbmnycmp` returns 1.

Usage

- `dbmnycmp` compares two DBMONEY values.
- 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.

Related Information

dbmny4cmp [page 217]

2.97 dbmnycopy

Copy a DBMONEY value.

Syntax

```
RETCODE dbmnycopy(dbproc, src, dest)

DBPROCESS    *dbproc;
DBMONEY        *src;
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.

src

A pointer to the source DBMONEY value.

dest

A pointer to the destination DBMONEY variable.

Returns

SUCCEED or FAIL.

`dbmnycopy` returns FAIL if either `<src>` or `<dest>` is NULL.

Usage

- `dbmnycopy` copies the `<src>` DBMONEY value to the `<dest>` DBMONEY value.
- 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.

Related Information

dbmnycopy [page 229]
dbmnyminus [page 240]
dbmny4minus [page 221]

2.98 dbmnydec

Decrement a DBMONEY value by one ten-thousandth of a dollar.

Syntax

```
RETCODE dbmnydec(dbproc, mnyptr)

DBPROCESS       *dbproc;
DBMONEY            *mnyptr;
```

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 decrement.

Returns

SUCCEED or FAIL.

`dbmnydec` returns FAIL in case of overflow or if `<mnyptr>` is NULL.

Usage

- `dbmnydec` decrements a DBMONEY value by one ten-thousandth of a dollar.
- 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 *`<mnyptr>` are undefined.
- 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.

Related Information

dbmnyinc [page 235]
dbmnymaxneg [page 238]

2.99 dbmnydivide

Divide one DBMONEY value by another.

Syntax

```
RETCODE dbmnydivide(dbproc, m1, m2, quotient)

DBPROCESS    *dbproc;
DBMONEY        *m1;
DBMONEY        *m2;
DBMONEY        *quotient;
```

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.

m1

A pointer to the DBMONEY value serving as dividend.

m2

A pointer to the DBMONEY value serving as divisor.

quotient

A pointer to a DBMONEY variable to hold the result of the division.

Returns

SUCCEED or FAIL.

`dbmnydivide` returns FAIL in case of overflow or division by zero, or if `<m1>`, `<m2>`, or `<quotient>` is NULL.

Usage

- dbmnydivide divides the <m1> DBMONEY value by the <m2> DBMONEY value and places the result in *<quotient>.
- In case of overflow or division by zero, dbmnydivide returns FAIL and sets *<quotient> to $0.0000.
- 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.

Related Information

dbmnyadd [page 226]
dbmnysub [page 249]
dbmnymul [page 241]
dbmnyminus [page 240]
dbmny4add [page 216]
dbmny4sub [page 224]
dbmny4mul [page 222]
dbmny4divide [page 220]
dbmny4minus [page 221]

2.100 dbmnydown

Divide a DBMONEY value by a positive integer.

Syntax

```
RETCODE dbmnydown(dbproc, mnyptr, divisor, remainder)

DBPROCESS    *dbproc;
DBMONEY        *mnyptr;
int                  divisor;
int                *remainder;
```

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 divide. *`<mnyptr>` will also contain the result of the division.

divisor

The integer by which *`<mnyptr>` will be divided. `<divisor>` must be positive, and must be less than or equal to 65535.

remainder

A pointer to an integer variable to hold the remainder from the division, in ten-thousandths of a dollar. If `<remainder>` is passed as NULL, no remainder is returned.

Returns

SUCCEED or FAIL.

`dbmnydown` returns FAIL if `<mnyptr>` is NULL, or if `<divisor>` is not between 1 and 65535.

Usage

- `dbmnydown` divides a DBMONEY value by a short integer and places the result back in the original DBMONEY variable.
- `dbmnydown` places the remainder of the division into *`<remainder>`. *`<remainder>` is an integer representing the number of ten-thousandths of a dollar left after the division.
- `<divisor>` must be greater than or equal to one and less than or equal to 65535.
- 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.

Related Information

dbmnyscale [page 247]
dbmnydivide [page 232]
dbmny4divide [page 220]

2.101 dbmnyinc

Increment a DBMONEY value by one ten-thousandth of a dollar.

Syntax

```
RETCODE dbmnyinc(dbproc, mnyptr)

DBPROCESS    *dbproc;
DBMONEY        *mnyptr;
```

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 increment.

Returns

SUCCEED or FAIL.

`dbmnyinc` returns FAIL in case of overflow or if `<mnyptr>` is NULL.

Usage

- `dbmnyinc` increments a DBMONEY value by one ten-thousandth of a dollar.
- An attempt to increment the maximum positive DBMONEY value will result in overflow. In case of overflow `dbmnyinc` returns FAIL. *`<mnyptr>` is undefined in this case.
- 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.

Related Information

dbmnydec [page 230]
dbmnymaxpos [page 239]

2.102 dbmnyinit

Prepare a DBMONEY value for calls to `dbmnyndigit`.

Syntax

```
RETCODE dbmnyinit(dbproc, mnyptr, trim, negative)

DBPROCESS    *dbproc;
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".

Returns

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 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 `dbmnyinit` reference page contains an example that demonstrates the use of `dbmnyinit`.

Related Information

dbmnyndigit [page 243]
dbmnyndigit [page 243]
dbconvert [page 124]
dbmnyndigit [page 243]

2.103 dbmnymaxneg

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.

Returns

SUCCEED or FAIL.

`dbmnymaxneg` returns FAIL if `<dest>` is NULL.

Usage

- `dbmnymaxneg` fills *`<dest>` with the maximum negative DBMONEY value supported.
- 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.

Related Information

dbmnymaxpos [page 239]

2.104 dbmnymaxpos

Return the maximum positive DBMONEY value supported.

Syntax

```
RETCODE dbmnymaxpos(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.

Returns

SUCCEED or FAIL.

`dbmnymaxpos` returns FAIL if `<dest>` is NULL.

Usage

- `dbmnymaxpos` fills *`<dest>` with the maximum positive DBMONEY value supported.

- 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.

Related Information

dbmnymaxneg [page 238]

2.105 dbmnyminus

Negate a DBMONEY value.

Syntax

```
RETCODE dbmnyminus(dbproc, src, dest)

DBPROCESS     *dbproc;
DBMONEY         *src;
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.

src

A pointer to a DBMONEY value.

dest

A pointer to a DBMONEY variable to hold the result of the negation.

Returns

SUCCEED or FAIL.

`dbmnyminus` returns FAIL in case of overflow, or if `<src>` or `<dest>` is NULL.

Usage

- `dbmnyminus` negates the `<src>` DBMONEY value and places the result into *`<dest>`.
- In case of overflow, `dbmnyminus` returns FAIL. *`<dest>` is undefined in this case. An attempt to negate the maximum negative DBMONEY value will result in overflow.
- 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.

Related Information

dbmny4minus [page 221]
dbmnycopy [page 229]
dbmny4copy [page 218]

2.106 dbmnymul

Multiply two DBMONEY values.

Syntax

```
RETCODE dbmnymul(dbproc, m1, m2, product)

DBPROCESS    *dbproc;
DBMONEY        *m1;
DBMONEY        *m2;
DBMONEY        *product;
```

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.

m1

A pointer to a DBMONEY value.

m2

A pointer to a DBMONEY value.

product

A pointer to a DBMONEY variable to hold the result of the multiplication.

Returns

SUCCEED or FAIL.

`dbmnymul` returns FAIL in case of overflow, or if `<m1>`, `<m2>`, or `<product>` is NULL.

Usage

- `dbmnymul` multiplies the `<m1>` DBMONEY value by the `<m2>` DBMONEY value and places the result in *`<product>`.
- In case of overflow, `dbmnymul` returns FAIL and sets *`<product>` to $0.0000.
- 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.

Related Information

dbmnyadd [page 226]
dbmnysub [page 249]
dbmnydivide [page 232]
dbmnyminus [page 240]
dbmny4add [page 216]
dbmny4sub [page 224]
dbmny4mul [page 222]
dbmny4divide [page 220]
dbmny4minus [page 221]

2.107 dbmnyndigit

Return the rightmost digit of a DBMONEY value as a DBCHAR.

Syntax

```
RETCODE dbmnyndigit(dbproc, mnyptr, value, zero)

DBPROCESS     *dbproc;
DBMONEY         *mnyptr;
DBCHAR            *value;
DBBOOL             *zero;
```

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 a DBMONEY value. Each call to `dbmnyndigit` divides this value by 10 and places the result back into *`<mnyptr>`.

value

A pointer to a DBCHAR variable to fill with the character representation of the rightmost digit of the DBMONEY value.

zero

A pointer to a DBBOOL variable. Each call to `dbmnyndigit` divides *`<mnyptr>` by 10 and puts the character representation of the remainder of the division in *`<value>`. If the result of the division is $0.0000, `dbmnyndigit` sets *`<zero>` to "true". Otherwise, *`<zero>` is set to "false". If `<zero>` is passed as NULL, this information is not returned.

Returns

SUCCEED or FAIL.

`dbmnyndigit` returns FAIL if `<mnyptr>` or `<value>` is NULL.

Usage

- `dbmnyndigit` returns the rightmost digit of a DBMONEY value as a DBCHAR.
- `dbmnyndigit` divides a DBMONEY value by 10. It places the character representation of the remainder of the division in *`<value>`, and replaces *`<mnyptr>` with the result of the division. If the result of the division is $0.0000, `dbmnyndigit` sets *`<zero>` to “true”.
- To get all the digits of a DBMONEY value, call `dbmnydigit` repeatedly, until *`<zero>` is “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.
- `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 `dbmnyndigit` and `dbmnyinit`:

```
/*
** This example demonstrates dbmnyinit() and
** dbmnyndigit(). 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(SUCCEED);
}
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';
}
```

Related Information

dbmnyinit [page 236]
dbconvert [page 124]
dbmnyinit [page 236]

2.108 dbmnyscale

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

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 multiply. *`<mnyptr>` will also contain the result of the `dbmnyscale` operation.

multiplier

The integer by which *`<mnyptr>` will be multiplied. `<multiplier>` must be positive, and must be greater than or equal to 1, and less than or equal to 65535.

addend

An integer representing the number of ten-thousandths of a dollar to add to *`<mnyptr>` after the multiplication.

Returns

SUCCEED or FAIL.

`dbmnyscale` returns FAIL if `<mnyptr>` is NULL, if overflow occurs, or if `<multiplier>` is not between 1 and 65535.

Usage

- `dbmnyscale` multiplies a DBMONEY value by a short integer, adds `<addend>` ten-thousandths of a dollar, and places the result back in the original DBMONEY variable.
- `<multiplier>` must be greater than or equal to 1, and less than or equal to 65535.
- In case of overflow, `dbmnyscale` returns FAIL. *`<mnyptr>` is undefined in this case.
- 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.

Related Information

dbmnydown [page 233]
dbmnymul [page 241]

dbmny4mul [page 222]

2.109 dbmnysub

Subtract one DBMONEY value from another.

Syntax

```
RETCODE dbmnysub(dbproc, m1, m2, difference)

DBPROCESS      *dbproc;
DBMONEY          *m1;
DBMONEY          *m2;
DBMONEY           *difference;
```

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.

m1

A pointer to the DBMONEY value to be subtracted from.

m2

A pointer to the DBMONEY value to subtract.

difference

A pointer to a DBMONEY variable to hold the result of the subtraction.

Returns

SUCCEED or FAIL.

`dbmnysub` returns FAIL in case of overflow, or if `<m1>`, `<m2>`, or `<difference>` is NULL.

Usage

- dbmnysub subtracts the <m2> DBMONEY value from the <m1> DBMONEY value and places the result in *<difference>.
- In case of overflow, dbmnysub returns FAIL and sets <difference> to $0.0000.
- 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.

Related Information

dbmnyadd [page 226]
dbmnymul [page 241]
dbmnydivide [page 232]
dbmnyminus [page 240]
dbmny4add [page 216]
dbmny4sub [page 224]
dbmny4mul [page 222]
dbmny4divide [page 220]
dbmny4minus [page 221]

2.110 dbmnyzero

Initialize a DBMONEY value to $0.0000.

Syntax

```
RETCODE dbmnyzero(dbproc, mnyptr)

DBPROCESS   *dbproc;
DBMONEY       *mnyptr;
```

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 initialize.

Returns

SUCCEED or FAIL.

`dbmnyzero` returns FAIL if `<mnyptr>` is NULL.

Usage

- `dbmnyzero` initializes a DBMONEY value to $0.0000.
- 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.

Related Information

dbmny4zero [page 225]

2.111 dbmonthname

Determine the name of a specified month in a specified language.

Syntax

```
char *dbmonthname(dbproc, language, monthnum,
                shortform)

DBPROCESS    *dbproc;
char                  *language;
int                      monthnum;
DBBOOL              shortform;
```

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.

language

The name of the desired language.

monthnum

The number of the desired month. Month numbers range from 1 (January) to 12 (December).

shortform

A Boolean value indicating whether the long or short form of the month name is desired. If `<shortform>` is “true”, `dbmonthname` returns the short form of the month name; if `<shortform>` is “false”, `dbmonthname` returns the full month name. For example, if the month name desired is the U.S. English short form for January, “Jan” is returned.

Short forms of month names are defined in localization files on a per-localization-file basis.

Returns

The name of the specified month on success; a NULL pointer on error.

Usage

- `dbmonthname` returns the name of the specified month in the specified language. If no language is specified (`<language>` is NULL), `<dbproc>`’s current language is used. If both `<language>` and `<dbproc>` are NULL, DB-Library’s default language (if any) is used.
- The following code fragment illustrates the use of `dbmonthname`:

```
for (monthnum = 1; monthnum <= 12; monthnum++)
printf("Month %d: %s\n", monthnum,
dbmonthname((DBPROCESS *)NULL,
char *)NULL, monthnum, TRUE),
dbmonthname((DBPROCESS *)NULL,
(char *)NULL, monthnum, FALSE));
```

Related Information

db12hour [page 62]

dbdateorder [page 163]
dbdayname [page 169]
DBSETLNATLANG [page 371]
dbsetopt [page 385]

2.112 DBMORECMDS

Indicate whether there are more commands to be processed.

Syntax

```
RETCODE DBMORECMDS(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.

Returns

SUCCEED or FAIL, indicating whether there are more results from the command batch.

Usage

- The application can use this macro to determine whether there are more results to process.
- `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`.
- Applications rarely need this routine, because they can simply call `dbresults` until it returns NO_MORE_RESULTS.

Related Information

DBCMDROW [page 112]
dbresults [page 320]
DBROWS [page 333]
DBROWTYPE [page 334]

2.113 dbmoretext

Send part of a `text` or `image` value to the server.

Syntax

```
RETCODE dbmoretext(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/server process. It contains all the information that DB-Library uses to manage communications and data between the front end and server.

size

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`.

text

A pointer to the `text` or `image` portion to be written.

Returns

SUCCEED or FAIL.

Usage

- This routine is used 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 long data buffers.
- `dbmoretext` and `dbwritetext` are used in updates only, and serve to replace the Transact-SQL `update` statement.
- `dbsqlok` and `dbresults` must be called before the first call to `dbmoretext` and after the last call to `dbmoretext`.
- 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 calls `dbsetopt` to make `<@@textsize>` larger.
  The DB-Library/C option DBTEXTLIMIT limits the size of `text` or `image` values that DB-Library/C reads.

Related Information

dbwritetext [page 443]
dbtxptr [page 431]
dbtxtimestamp [page 433]
dbwritetext [page 443]
dbwritetext [page 443]

2.114 dbmsghandle

Install a user function to handle server messages.

Syntax

```
int  (*dbmsghandle(handler))()
int                (*handler)();
```

Parameters

handler

A pointer to the user function that is called whenever DB-Library receives an error or informational message from the server. DB-Library calls this function with eight parameters listed as:

Message handler parameters

Parameter	Meaning
`<dbproc>`	The affected DBPROCESS.
`<msgno>`	The current message's number (datatype DBINT). These numbers are documented in the `sysmessages` table.
`<msgstate>`	The current message's error state number (datatype `<int>`). These numbers provide SAP Technical Support with information about the context of the error.
`<severity>`	The current message's information class or error severity (datatype `<int>`). These numbers are documented in the SAP Adaptive Server Enterprise documentation.
`<msgtext>`	The null-terminated text of the current message (datatype `<char>` *).
`<srvname>`	The null-terminated name of the server that generated the message (datatype `<char>` *). A server's name is stored in the `srvname` column of its `sysservers` 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, `<srvname>` is of length of 0.
`<procname>`	The null-terminated name of the stored procedure that generated the message (datatype `char` *). If the message was not generated by a stored procedure, `<procname>` is of length of 0.
`<line>`	The number of the command batch or stored procedure line that generated the message (datatype `<int>`). 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 `<line>` is 3. `< line>` is 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 `dbrpcsend`) 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 <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);
}
```

Returns

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 `dbsqlexec`.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.
- 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 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.
- provides information on when DB-Library/C calls an application's message and error handlers.

When DB-Library calls message and error handlers

Error or message	**Message handler called?**	**Error handler called?**
SQL syntax error	Yes	Yes (SYBESMSG). (Code the handler to ignore the message.)
SQL `print` statement	Yes	No.
SQL `raiserror`	Yes	No.
Server dies	No	Yes (SYBESEOF). (Code your handler to exit the application.)
Timeout from the server	No	Yes (SYBETIME). (To wait for another timeout period, code your handler to return -INT_CONTINUE.)
Deadlock on query	Yes	No. (Code your handler to test for deadlock.)
Timeout on login	No	Yes (SYBEFCON).
Login fails (`dbopen`)	Yes	Yes (SYBEPWD). (Code your handler to exit the application.)
Use database message	Yes (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).

Related Information

dbsetuserdata [page 390]
dberrhandle [page 172]
dbgetuserdata [page 200]
dbsetuserdata [page 390]

2.115 dbname

Return the name of the current database.

Syntax

```
char *dbname(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.

Returns

A pointer to the null-terminated name of the current database.

Usage

- `dbname` returns the name of the current database.
- If you need to keep track of when the database changes, use `dbchange`.

Related Information

dbchange [page 104]
dbuse [page 437]

2.116 dbnextrow

Read the next result row into the row buffer and into any program variables that are bound to column data.

Syntax

```
STATUS dbnextrow(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.

Returns

`dbnextrow` returns:

- REG_ROW if a regular row has been read. A regular row is any row that matches the query's `where` clause.
- A `<computeid>` if a compute row was read. A compute row is a row that is generated by a `compute` clause. The `<computeid>` matches the number of the compute row that was read; the first compute row is 1, the second is 2, and so on. A `<computeid>` cannot match any other of the return types for this function.
- 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 is 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` returns 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.

Usage

- `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 are 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 */
dbsqlexec(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`.

Related Information

dbaltbind [page 68]

dbbind [page 88]
dbcanquery [page 102]
dbclrbuf [page 107]
dbgetrow [page 197]
dbprrow [page 284]
dbsetrow [page 387]
Options [page 466]

2.117 dbnpcreate

Create a notification procedure.

Syntax

```
RETCODE dbnpcreate(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/C uses to manage communications and data between the front end and the server.

Returns

SUCCEED or FAIL.

Usage

- `dbnpcreate` creates a notification procedure. A notification procedure is a special type of SAP Open Server registered procedure. A notification procedure differs from a normal SAP Open Server registered procedure in that it contains no executable statements. Notification procedures are the only type of SAP Open Server registered procedure that a DB-Library/C application can create.

- The notification procedure name and its parameters must have been previously defined using `dbnpdefine` and `dbregparam`.
- 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`
- 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 SAP Open Server, some of which may be notification procedures.
- 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.
- Notification procedures may be created only in SAP Open Server. At this time, Adaptive Server Enterprise does not support notification procedures.
- 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 creating a notification procedure:

```
 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");
}
```

Related Information

dbreginit [page 306]
dbregparam [page 311]
dbregwatch [page 315]
dbregnowatch [page 309]

2.118 dbnpdefine

Define a notification procedure.

Syntax

```
RETCODE dbnpdefine(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 notification procedure being defined.

namelen
: The length of `<procedure_name>`, in bytes. If `<procedure_name>` is null-terminated, pass `<namelen>` as DBNULLTERM.

Returns

SUCCEED or FAIL.

Usage

- `dbnpdefine` defines a notification procedure. Defining a notification procedure is the first step in creating it.
- A notification procedure is a special type of SAP Open Server registered procedure. A notification procedure differs from a normal SAP Open Server registered procedure in that it contains no executable statements. Notification procedures are the only type of SAP Open Server registered procedure that a DB-Library/C application can create.
- 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`
- 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 SAP Open Server, some of which may be notification procedures.
- This is an example of defining 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
** 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");
}
```

Related Information

dbregparam [page 311]
dbnpcreate [page 262]
dbreglist [page 308]

2.119 dbnullbind

Associate an indicator variable with a regular result row column.

Syntax

```
RETCODE dbnullbind(dbproc, column, indicator)

DBPROCESS     *dbproc;
int                         column;
DBINT                  *indicator;
```

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 that is to be associated with the indicator variable.

indicator
: A pointer to the indicator variable.

Returns

SUCCEED or FAIL.

`dbnullbind` returns FAIL if `<column>` is invalid.

Usage

- `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.
- The indicator variable is set when regular result rows are processed using `dbnextrow`. The possible values are:
    - -1 if the column is NULL.
    - The full length of column's data, in bytes, if `<column>` 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 `<column>`'s data.
    - 0 if `<column>` was bound and copied successfully to a program variable.

Note

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

Related Information

dbanullbind [page 87]
dbbind [page 88]
dbdata [page 151]
dbdatlen [page 168]
dbnextrow [page 260]

2.120 dbnumalts

Return the number of columns in a compute row.

Syntax

```
int dbnumalts(dbproc, computeid)

DBPROCESS    *dbproc;
int                          computeid;
```

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`.

Returns

The number of columns for the particular computeid. `dbnumalts` returns -1 if `<computeid>` is invalid.

Usage

`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(<dbproc>`, 1) returns 3:

```
select dept, year, sales from employee
```

```
order by dept, year
```

```
compute avg(sales), min(sales),
```

```
max(sales) by dept
```

Related Information

dbadata [page 64]
dbadlen [page 66]
dbaltlen [page 81]
dbalttype [page 84]
dbgetrow [page 197]
dbnextrow [page 260]
dbnumcols [page 269]

2.121 dbnumcols

Determine the number of regular columns for the current set of results.

Syntax

```
int dbnumcols(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.

Returns

The number of columns in the current set of results. If there are no columns, `dbnumcols` returns 0.

Usage

- `dbnumcols` returns the number of regular (that is, non-compute) columns in the current set of results.
- Here is a program fragment that illustrates the use of `dbnumcols`:

```
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
*/
dbsqlexec(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");
}
```

Related Information

dbcollen [page 115]
dbcolname [page 116]
dbnumalts [page 267]

2.122 dbnumcompute

Return the number of `compute` clauses in the current set of results.

Syntax

```
int dbnumcompute(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.

Returns

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(<dbproc>)` will return 2 since there are two `compute` clauses in the `select` statement:

```
select dept, name from employee
```

```
order by dept, name
```

```
compute count(name) by dept
```

```
compute count(name)
```

Related Information

dbnumalts [page 267]
dbresults [page 320]

2.123 DBNUMORDERS

Return the number of columns specified in a Transact-SQL `select` statement's `order by` clause.

Syntax

```
int DBNUMORDERS(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.

Returns

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.

Related Information

dbordercol [page 278]

2.124 dbnumrets

Determine the number of return parameter values generated by a stored procedure.

Syntax

```
int dbnumrets(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.

Returns

The number of return parameter values associated with the most recently executed stored procedure.

Usage

- `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.
- 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 `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 a `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 *SAP 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 dbretdata [page 322].

Related Information

dbretdata [page 322]
dbnextrow [page 260]
dbresults [page 320]
dbretdata [page 322]
dbretlen [page 325]

dbretname [page 327]
dbrettype [page 331]
dbrpcinit [page 335]
dbrpcparam [page 337]

2.125 dbopen

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 is 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*.

i 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.

Returns

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 is aborted.

Related Information

Usage for dbopen [page 275]

dbsetifile [page 361]

dbclose [page 106]

dbexit [page 176]

dbinit [page 203]

dblogin [page 213]

dbloginfree [page 215]

dbsetifile [page 361]

dbuse [page 437]

2.125.1 Usage for dbopen

Review how to use dbopen.

- 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` attempts 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 attempts to connect with VENUS. If it fails to connect with VENUS, it attempts 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` waits at the top of the loop before going through the list again. These numbers are optional. If they are not included, `dbopen` tries 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 returns 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.

> i 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. Note that an application can set or retrieve the maximum number of DBPROCESS structures with `dbsetmaxprocs` and `dbgetmaxprocs`.
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.

2.126 dbordercol

Return the id of a column appearing in the most recently executed query's order by clause.

Syntax

```
int dbordercol(dbproc, order)

DBPROCESS   *dbproc;
int                        order;
```

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.

order

The id that identifies the particular order by column of interest. The first column named within the order by clause is number 1.

Returns

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.

Usage

This routine returns the id of the column that appears in a specified location within the order by clause of a SQL select command.

For example, in given the SQL statement, the call dbordercol(<dbproc>, 1) will return 3 since the first column named in the order by clause refers to the third column in the query's select list:

```
select dept, name, salary from employee
```

```
order by salary, name
```

Related Information

DBNUMORDERS [page 271]

2.127 dbpoll

Verifies that a server response has arrived for a DBPROCESS.

Syntax

```
RETCODE dbpoll(dbproc, milliseconds, ready_dbproc,
             return_reason)

DBPROCESS     *dbproc;
long                      milliseconds;
DBPROCESS    **ready_dbproc;
int                          *return_reason;
```

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 server.

`<dbproc>` represents the DBPROCESS connection that `dbpoll` checks.

If `<dbproc>` is passed as NULL, `dbpoll` checks all open DBPROCESS connections to see if a response has arrived for any of them.

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` does 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.

i 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 is 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.

i 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.

Returns

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.

Related Information

Usage for dbpoll [page 281]

DBIORDESC [page 203]

DBRBUF [page 291]

dbresults [page 320]
dbreghandle [page 303]
dbsqlok [page 404]

2.127.1 Usage for dbpoll

Review the usage for dbpoll.

- `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` checks. 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 `dbrpcsend`) 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 dbsqlok [page 404]`dbsqlok`.

  i Note

  On occasion `dbpoll` may report that data is ready for `dbsqlok` to read when only the first bytes of the server response are present. When this occurs, `dbsqlok` 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 are blocked. 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 DBPROCESSes, 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 is not 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 processan 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 */
}
```

Related Information

dbsqlok [page 404]

2.128 dbprhead

Print the column headings for rows returned from the server.

Syntax

```
void dbprhead(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.

Returns

None.

Usage

- 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 `dbprrow`.
- 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.

Related Information

dbbind [page 88]
dbnextrow [page 260]
dbprrow [page 284]
dbresults [page 320]
dbspr1row [page 395]
dbsprhead [page 398]
dbsprline [page 400]

2.129 dbprrow

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.

Returns

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.

Related Information

dbbind [page 88]
dbnextrow [page 260]
dbprhead [page 283]
dbresults [page 320]
dbspr1row [page 395]
dbsprhead [page 398]
dbsprline [page 400]

2.130 dbprtype

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`, `dbalttype`, `dbrettype`, or `dbaltop`.

Returns

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`, `dbalttype`, `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` takes a `dbcoltype` token value representing the server binary datatype (SYBBINARY) and return the string "binary."
- A list provides of the token strings that `dbprtype` can return and their token value equivalents:

Token values and their string equivalents

Token string	Token value	Description
char	SYBCHAR	`char` datatype
text	SYBTEXT	`text` datatype

Token string	Token value	Description
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

Related Information

dbaltop [page 82]
dbalttype [page 84]
dbcoltype [page 119]

dbrettype [page 331]
Types [page 470]

2.131 dbqual

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

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.

tabnum
The number of the table of interest, as specified in the select statement's from clause. Table numbers start at 1. If <tabnum> is -1, the <tabname> parameter is used to identify the table.

tabname
A pointer to the null-terminated name of a table specified in the select statement's from clause. <tabname> is ignored unless <tabnum> is passed as -1.

Returns

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 dbfreequal.

dbqual returns 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.

dbqual also returns a NULL pointer if the preceding select did not include the for browse option.

Usage

- `dbqual` is one of the DB-Library browse mode routines. See Browse Mode [page 44] 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. 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`).
  The `where` clause produced by `dbqual` begins with the keyword `where` 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.
  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 `dbqual`-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:

```
                /* This code fragment illustrates a technique for
** 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.
*/
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 ((dbsqlexec(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
Server Enterprise
** 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 called after `dbnextrow`.
- For a complete example that uses `dbqual` to perform a browse mode update, see the sample programs included with DB-Library.

Related Information

dbcolbrowse [page 113]

dbcolsource [page 118]

dbfreequal [page 184]

dbtabbrowse [page 419]

dbtabcount [page 420]

dbtabname [page 422]

dbtabsource [page 423]

dbtsnewlen [page 427]

dbtsnewval [page 428]

dbtsput [page 430]

Browse Mode [page 44]

2.132 DBRBUF

(UNIX only) Determine whether the DB-Library network buffer contains any unread bytes.

Syntax

```
DBBOOL DBRBUF(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.

Returns

"TRUE" (bytes remain in buffer) or "FALSE" (no bytes in buffer).

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.

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.

Usage

- This routine lets the application know if the DB-Library network buffer contains any bytes yet unread.
- DBRBUF is ordinarily used in conjunction with `dbsqlok` and `DBIORDESC`.
- `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.

- 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`.
- 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`.

Related Information

DBIORDESC [page 203]
DBIORDESC [page 203]
dbpoll [page 279]
dbsqlok [page 404]
dbresults [page 320]

2.133 dbreadpage

Read a page of binary data from the server.

Syntax

```
DBINT dbreadpage(dbproc, dbname, pageno, buf)

DBPROCESS    *dbproc;
char                   *dbname;
DBINT                 pageno;
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 read.

buf

A pointer to a buffer to hold the received page data. SAP Adaptive Server Enterprise pages are currently 2048 bytes long.

Returns

The number of bytes read from the server. If the operation was unsuccessful, `dbreadpage` returns -1.

Usage

- `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.
- dbreadpage alters the contents of the DBPROCESS command buffer.

⚠ Caution

Use this routine only if you are absolutely sure that you know what you are doing!

Related Information

dbmsghandle [page 255]
dbwritepage [page 442]

2.134 dbreadtext

Read part of a `text` or `image` value from the server.

Syntax

```
STATUS dbreadtext(dbproc, buf, bufsize)

DBPROCESS     *dbproc;
```

```
void                     *buf;
DBINT                   bufsize;
```

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.

buf

A pointer to a caller-allocated buffer that will contain the chunk of `text` or `image` data.

bufsize

The size of the caller's buffer, in bytes.

Returns

The following table lists the return values for `dbreadtext`:

`dbreadtext` **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

- `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.
- To read successive chunks of the same SYBTEXT or SYBIMAGE value, call `dbreadtext` until it returns 0 (end of row).
- Use `dbreadtext` in place of `dbnextrow` to read SYBTEXT and SYBIMAGE values.
- `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.
- 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);
}
}
}
```

Related Information

dbmoretext [page 254]
dbnextrow [page 260]
dbwritetext [page 443]

2.135 dbrecftos

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.

Returns

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.

Related Information

dbopen [page 274]

2.136 dbrecvpassthru

Receive a TDS packet from a server.

Syntax

```
RETCODE dbrecvpassthru(dbproc, recv_bufp)

DBPROCESS     *dbproc;
DBVOIDPTR       *recv_bufp;
```

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.

recv_bufp

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.

Returns

DB_PASSTHRU_MORE, DB_PASSTHRU_EOM, or FAIL.

Usage

- `dbrecvpassthru` receives a TDS (Tabular Data Stream) packet from a server.
- 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.
- `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.
- `dbrecvpassthru` reads a packet of bytes from the server connection identified by `<dbproc>` and sets *`<recv_bufp>` to point to the buffer containing the bytes.

- 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.
- 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.
- 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;
}
```

Related Information

dbgetpacket [page 196]
DBSETLPACKET [page 377]
dbsendpassthru [page 349]

2.137 dbregdrop

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.

Returns

SUCCEED, DBNOPROC, or FAIL.

Usage

- `dbregdrop` drops a registered procedure from SAP 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 SAP 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 SAP Open Server, `dbregdrop` returns DBNOPROC. An application can use `dbreglist` to obtain a list of registered procedures currently defined in SAP 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);
 }
```

Related Information

dbnpcreate [page 262]

dbreglist [page 308]

2.138 dbregexec

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, SAP Open Server notifies all DBPROCESSes watching for the execution of this registered procedure.

If `options` is DBNOTIFYNEXT, SAP Open Server notifies only the DBPROCESS that has been watching the longest.

Returns

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 SAP Open Server. At this time, Adaptive Server Enterprise does not support registered procedures. An application can use `dbnpcreate`, `dbregparam`, and `dbnpcreate` 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");
}
```

Related Information

dbreginit [page 306]
dbregparam [page 311]
dbregwatch [page 315]
dbregnowatch [page 309]

2.139 dbreghandle

Install a handler routine for a registered procedure notification.

Syntax

```
RETCODE dbreghandle(dbproc, procedure_name, namelen,
                  handler)

DBPROCESS      *dbproc;
DBCHAR             *procedure_name;
DBSMALLINT     namelen;
INTFUNCPTR     handler;
```

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 for which the handler is being installed.

If `<procedure_name>` is passed as NULL, the handler is installed as a default handler. The default handler is called for all registered procedure notifications read by this DBPROCESS connection for which no other handler has been installed.

namelen

The length of `<procedure_name>`, in bytes. If `<procedure_name>` is null-terminated, pass `<namelen>` as DBNULLTERM.

handler

A pointer to the function to be called by DB-Library/C when the registered procedure notification is read.

If `<handler>` is passed as NULL, any handler previously installed for the registered procedure is uninstalled.

Returns

SUCCEED or FAIL.

Usage

- `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.
  Because a notification procedure is simply a special type of registered procedure, a handler for a notification procedure may also be installed using `dbreghandle`.
- An application receives an asynchronous notification only if it has previously called `dbregwatch` with `<options>` passed as DBNOWAITONE or DBNOWAITALL. This call tells SAP Open Server that the application is interested in the execution of the registered procedure, that it receives the notification asynchronously, and reads the notification through a particular DBPROCESS connection.
- If no handler is installed for a notification, DB-Library/C raises an error when the DBPROCESS connection reads the notification.
- 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 is 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 is called before `dbpoll` returns.

    - If the DBPROCESS connection is sending commands, the notification is read and the notification handler called during `dbsqlexec` or `dbsqlok`. After the notification handler returns, the flow of control continues.
    - 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, the flow of control continues.
    - 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 `dbcanquery`. 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 following example. 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);
  }
```

Related Information

dbregwatch [page 315]
dbregnowatch [page 309]
dbregparam [page 311]
dbregexec [page 301]

2.140 dbreginit

Initiate execution of a registered procedure.

Syntax

```
RETCODE dbreginit(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 server.

procedure_name
: A pointer to the name of the registered procedure being executed.

namelen
: The length of `<procedure_name>`, in bytes. If `<procedure_name>` is null-terminated, pass `<namelen>` as DBNULLTERM.

Returns

SUCCEED or FAIL.

Usage

- `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`.
- 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 using `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, 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");
}
```

Related Information

dbregparam [page 311]
dbregexec [page 301]
dbregwatch [page 315]
dbreglist [page 308]
dbregwatchlist [page 318]

2.141 dbreglist

Return a list of registered procedures currently defined in SAP Open Server solution.

Syntax

```
RETCODE dbreglist(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/C uses to manage communications and data between the front end and the server.

Returns

SUCCEED or FAIL.

Usage

- `dbreglist` returns a list of registered procedures currently defined in SAP Open Server. Because a notification procedure is simply a special type of registered procedure, notification procedures are 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 */
```

Related Information

dbregwatchlist [page 318]

dbregwatch [page 315]

2.142 dbregnowatch

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

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 is no longer interested in.

namelen

The length of `<procedure_name>`, in bytes. If `<procedure_name>` is null-terminated, pass `<namelen>` as DBNULLTERM.

Returns

SUCCEED, DBNOPROC, or FAIL.

Usage

- `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.
- It is meaningful to call `dbregnowatch` only if the DBPROCESS connection has previously requested an asynchronous notification using `dbregwatch`.
- If the procedure referenced by `<procedure_name>` is not defined in SAP Open Server, `dbregnowatch` 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 through `dbregwatchlist`.
- This is an example of canceling a request to be notified:

```
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");
}
```

Related Information

dbregwatch [page 315]
dbregwatchlist [page 318]
dbreghandle [page 303]
dbregexec [page 301]

2.143 dbregparam

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 expects 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.

Returns

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.

  i 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 SAP 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");
}
```

Related Information

dbnpdefine [page 264]

dbnpcreate [page 262]

dbreginit [page 306]

dbregexec [page 301]

dbnpdefine [page 264]

dbnpcreate [page 262]

dbregwatch [page 315]

2.144 dbregwatch

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 SAP 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` does 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 receives an asynchronous notification when the registered procedure executes. The connection receives only a single notification, even if the registered procedure executes multiple times.

If `options` is passed as DBNOWAITALL, `dbregwatch` returns immediately. The DBPROCESS connection receives an asynchronous notification when the registered procedure executes. The connection continues to receive notifications, one for each execution of the registered procedure, until it informs Open Server that it no longer wishes to receive them.

Returns

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` does not return until the DBPROCESS connection reads the notification that the registered procedure has executed.
    Open Server sends 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 does 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 sends only a single notification, even if the registered procedure executes multiple times.
    If `<options>` is DBNOWAITALL, Open Server continues 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 is called before `dbpoll` returns.
  - If the DBPROCESS connection is sending commands, the notification is read and the notification handler called during `dbsqlexec` 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.
- 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 asychronous
** 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 */
}
```

Related Information

dbpoll [page 279]

dbregexec [page 301]

dbregparam [page 311]

dbreglist [page 308]

dbregwatchlist [page 318]

dbregnowatch [page 309]

2.145 dbregwatchlist

Return a list of registered procedures that a DBPROCESS is watching for.

Syntax

```
RETCODE dbregwatchlist(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.

Returns

SUCCEED or FAIL.

Usage

- `dbregwatchlist` 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 `dbregwatchlist` includes notification procedures.
- The list of registered procedures is returned as rows that an application must explicitly process after calling `dbregwatchlist`. 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.
- The following code fragment illustrates how `dbregwatchlist` might be used in an application:

```
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 */
```

Related Information

dbregwatch [page 315]
dbresults [page 320]
dbnextrow [page 260]

2.146 dbresults

Set up the results of the next query.

Syntax

```
RETCODE dbresults(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.

Returns

SUCCEED, FAIL or NO_MORE_RESULTS.

`dbresults` 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.

The number of commands in the command buffer determines whether `dbsqlexec` or `dbresults` traps a runtime error. If the buffer contains only a single command, a runtime error causes `dbsqlexec` to fail. If the command buffer contains multiple commands, a runtime error does not cause `dbsqlexec` to fail. Instead, the `dbresults` call that processes the command causing the runtime error fails.

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 does 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` still returns 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 `dbsqlexec` or `dbsqlok` returns SUCCEED. The first call to `dbresults` always returns either SUCCEED or NO_MORE_RESULTS if the call to `dbsqlexec` or `dbsqlok` 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.

  > i Note
  >
  > All Transact-SQL commands are considered commands by `dbresults`. For a list of Transact-SQL commands, see the *SAP 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 `dbsqlexec` 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 */
dbsqlexec(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.

Related Information

dbrpcsend [page 339]
dbbind [page 88]
dbcancel [page 101]
dbnextrow [page 260]
dbpoll [page 279]
DBRBUF [page 291]
dbretstatus [page 329]
DBROWS [page 333]
dbrpcsend [page 339]
dbsqlexec [page 402]
dbsqlok [page 404]

2.147 dbretdata

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.

Returns

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 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 `dbsqlexec` 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 following routines 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 following fragment 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);
}
```

Related Information

dbnextrow [page 260]

dbnumrets [page 272]

dbresults [page 320]

dbretlen [page 325]

dbretname [page 327]

dbrettype [page 331]

dbrpcinit [page 335]

dbrpcparam [page 337]

2.148 dbretlen

Determine the length of a return parameter value generated by a stored procedure.

Syntax

```
DBINT dbretlen(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.

Returns

The length of the specified return parameter value. If `<retnum>` is out of range, `dbretlen` returns -1. If the return value is null, `dbretlen` returns 0.

Usage

- `dbretlen` returns the length of a particular return parameter value generated by a stored procedure. It is useful 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 `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 dbretdata [page 322].

Related Information

dbretdata [page 322]
dbnextrow [page 260]
dbnumrets [page 272]
dbresults [page 320]
dbretdata [page 322]
dbretname [page 327]
dbrettype [page 331]
dbrpcinit [page 335]
dbrpcparam [page 337]

2.149 dbretname

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

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.

Returns

A pointer to the null-terminated parameter name for the specified return value. If `<retnum>` is out of range, `dbretname` returns NULL.

Usage

- `dbretname` returns a pointer to the null-terminated parameter name associated with a return parameter value from a stored procedure. It is useful 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 `dbretname` 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 `dbretname` 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.
    - `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 dbretdata [page 322].

Related Information

dbretdata [page 322]
dbnextrow [page 260]
dbnumrets [page 272]
dbresults [page 320]
dbretdata [page 322]
dbretlen [page 325]
dbrettype [page 331]
dbrpcinit [page 335]
dbrpcparam [page 337]

2.150 dbretstatus

Determine the stored procedure status number returned by the current command or remote procedure call.

Syntax

```
DBINT dbretstatus(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.

Returns

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 return a status number of 0. For a list of return status numbers, see the *SAP 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 `dbsqlexec` 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 dbhasretstat [page 201].

Related Information

dbhasretstat [page 201]
dbhasretstat [page 201]
dbhasretstat [page 201]
dbnextrow [page 260]
dbresults [page 320]
dbretdata [page 322]
dbrpcinit [page 335]
dbrpcparam [page 337]
dbrpcsend [page 339]

2.151 dbrettype

Determine the datatype of a return parameter value generated by a stored procedure.

Syntax

```
int dbrettype(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.

Returns

A token value for the datatype of the specified return value.

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.
- SYBFLTN is returned as SYBFLT8.
- SYBINTN is returned as SYBINT1, SYBINT2, or SYBINT4, depending on the actual type of the SYBINTN.
  If `<retnum>` is out of range, -1 is returned.

Usage

- `dbrettype` returns the datatype of a return parameter value generated by a stored procedure. It is useful 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 `dbrettype` 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 `dbrettype` 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.
- `dbrettype` 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 [page 470].
- 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 dbretdata [page 322].

Related Information

dbprtype [page 286]
Types [page 470]
dbretdata [page 322]
dbnextrow [page 260]
dbnumrets [page 272]
dbprtype [page 286]
dbresults [page 320]
dbretdata [page 322]
dbretlen [page 325]

dbretname [page 327]
dbrpcinit [page 335]
dbrpcparam [page 337]

2.152 DBROWS

Indicate whether the current command actually returned rows.

Syntax

```
RETCODE DBROWS(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.

Returns

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`).

Related Information

DBCMDROW [page 112]
dbnextrow [page 260]
dbresults [page 320]
DBROWTYPE [page 334]

2.153 DBROWTYPE

Return the type of the current row.

Syntax

```
STATUS DBROWTYPE(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.

Returns

Three different types of values can be returned:

- 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. (For more information on the `<computeid>` variable, see dbaltbind [page 68].)
- If no rows have been read, or if the routine failed for any reason, NO_MORE_ROWS is returned.

Usage

- 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.

Related Information

dbaltbind [page 68]
dbnextrow [page 260]

2.154 dbrpcinit

Initialize a remote procedure call.

Syntax

```
RETCODE dbrpcinit(dbproc, rpcname, options)

DBPROCESS    *dbproc;
char                  *rpcname;
DBSMALLINT      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 uses to manage communications and data between the front end and the server.

rpcname
: 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.

Returns

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 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 have 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`.

Related Information

dbnextrow [page 260]
dbresults [page 320]
dbretdata [page 322]
dbretstatus [page 329]
dbrpcparam [page 337]
dbrpcsend [page 339]
dbsqlok [page 404]

2.155 dbrpcparam

Add a parameter to a remote procedure call.

Syntax

```
RETCODE dbrpcparam(dbproc, paramname, status, type,
                 maxlen, datalen, value)

DBPROCESS   *dbproc;
char                  *paramname;
BYTE                 status;
int                       type;
DBINT                maxlen;
DBINT                datalen;
BYTE                 *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 the server.

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 `dbrpcparam` 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* 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 is 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*. For types that have no C equivalent, such as SYBDATETIME, SYBMONEY, SYBNUMERIC, or SYBDECIMAL, use *dbconvert_ps* to initialize `<*value>`.

> **i 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.

Returns

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). See the reference page for `dbrpcinit` for a discussion of the differences between these techniques.
- 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 have 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 <type> is SYBCHAR, SYBVARCHAR, SYBBINARY, SYBVARBINARY, SYBBOUNDARY, and 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 SYBNUMERIC or SYBDECIMAL, use dbconvert_ps to initialize the DBNUMERIC or DBDECIMAL value in <*value> and specify its precision and scale.
- 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.

Related Information

Types [page 470]
Types [page 470]
dbconvert_ps [page 128]
dbrpcinit [page 335]
dbconvert_ps [page 128]
dbnextrow [page 260]
dbresults [page 320]
dbretdata [page 322]
dbretstatus [page 329]
dbrpcinit [page 335]
dbrpcsend [page 339]
dbsqlok [page 404]

2.156 dbrpcsend

Signal the end of a remote procedure call.

Syntax

```
RETCODE dbrpcsend(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.

Returns

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). See the reference page for `dbrpcinit` for a discussion of the differences between these techniques.
- 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`.

Related Information

dbrpcinit [page 335]
dbnextrow [page 260]
dbresults [page 320]
dbretdata [page 322]
dbretstatus [page 329]
dbrpcinit [page 335]
dbrpcparam [page 337]
dbsqlok [page 404]

2.157 dbrpwclr

Clear all remote passwords from the LOGINREC structure.

Syntax

```
void dbrpwclr(loginrec)

LOGINREC      *loginrec;
```

Parameters

loginrec

A pointer to a LOGINREC structure. This pointer serves as an argument to `dbopen`. You can allocate a LOGINREC structure by calling `dblogin`.

Returns

None.

Usage

- 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.
  `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.
- A single LOGINREC can be used repeatedly, in successive `dbopen` calls to different servers. `dbrpwclr` allows the application to remove any remote password information currently in the LOGINREC, so that successive calls to `dbopen` can contain different remote password information (specified with `dbrpwset`).

Related Information

dblogin [page 213]

dbopen [page 274]
dbrpwset [page 342]
DBSETLAPP [page 365]
DBSETLHOST [page 369]
DBSETLPWD [page 378]
DBSETLUSER [page 380]

2.158 dbrpwset

Add a remote password to the LOGINREC structure.

Syntax

```
RETCODE dbrpwset(loginrec, srvname, password, pwlen)

LOGINREC     *loginrec;
char                 *srvname;
char                 *password;
int                      pwlen;
```

Parameters

loginrec

A pointer to a LOGINREC structure. This pointer serves as an argument to `dbopen`. You can allocate a LOGINREC structure by calling `dblogin`.

srvname

The name of a server. A server's name is stored in the `<srvname>` column of its `sysservers` system table. When the first server calls a stored procedure located on the server designated by `<srvname>`, it uses the specified password to log in. If `<srvname>` is NULL, the specified password is considered a "universal" password, to be used with any server that does not have a password explicitly specified for it.

password

The password that the first server uses to log in to the specified server.

pwlen

The length of the password in bytes.

Returns

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

- 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.
  `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.
- 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.
- `dbrpwclr` clears all remote passwords from the LOGINREC.

Related Information

dblogin [page 213]
dbopen [page 274]
dbrpwclr [page 341]
DBSETLAPP [page 365]
DBSETLHOST [page 369]
DBSETLPWD [page 378]
DBSETLUSER [page 380]

2.159 dbsafestr

Double the quotes in a character string.

Syntax

```
RETCODE dbsafestr(dbproc, src, srclen, dest, destlen,
                quotetype)

DBPROCESS     *dbproc;
char                    *src;
DBINT                 srclen;
char                    *dest;
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 `<src>`, in bytes. If `<srclen>` is -1, `<src>` is assumed to be null-terminated.

dest

A pointer to a programmer-supplied buffer to contain the resulting string. `<dest>` 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 `<destlen>` is -1, `<dest>` is assumed to be large enough to hold the resulting string.

quotetype

The type of quotes to double.

Values for quotetype below lists the possible values for `<quotetype>`:

Values for quotetype

Value of `<quotetype>`	`dbsafestr`
DBSINGLE	Doubles all single quotes (') in `<src>`
DBDOUBLE	Doubles all double quotes (") in `<src>`
DBBOTH	Doubles all single and double quotes in `<src>`

Returns

SUCCEED or FAIL.

`dbsafestr` fails if the resulting string is too large for `<dest>`, or if an invalid `<quotetype>` is specified.

Usage

- `dbsafestr` doubles the single and/or double quotes found in a character string. This is useful when specifying literal quotes within a character string.

Related Information

dbcmd [page 110]
dbfcmd [page 177]

2.160 dbsechandle

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 the following table:

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 calls whenever the corresponding type of secure login is to be handled.

If `<handler>` is NULL and `<type>` is DBENCRYPT, DB-Library uses its default encryption handler.

If `<handler>` is NULL and `<type>` is DBLABELS, `dbsechandle` uninstalls any current label handler.

Returns

SUCCEED or FAIL.

Related Information

Usage for dbsechandle [page 346]
DBSETLENCRYPT [page 368]
dbopen [page 274]

2.160.1 Usage for dbsechandle

Review how to use `dbsechandle`.

- `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 calls when encrypting user passwords.
- DB-Library performs password encryption only if `DBSETLENCRYPT` has been called prior to calling `dbopen`.
- DB-Library calls its default encryption handler if a user function has not been installed.
- Typically, a user function is not installed for password encryption. This is because DB-Library's default encryption handler allows an application to perform password encryption when connecting to an SAP Adaptive Server Enterprise.
- A user-defined encryption handler should be installed by applications that are gateways. The encryption handler is 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 following example. 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,
  pwdlen, enc_key, keylen, outbuf, buflen, outlen)
  DBPROCESS *dbproc;
  BYTE *pwd;
  DBINT pwdlen;
  BYTE *enc_key;
  DBINT keylen;
  BYTE *outbuf;
  DBINT buflen;
  DBINT *outlen;
  ```

  where:
  - `<dbproc>` is the DBPROCESS.
  - `<pwd>` is the user password to be encrypted.
  - `<pwdlen>` 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 is 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 aborts the connection attempt.

Security label secure logins

- If type is DBLABELS, `dbsechandle` installs a function that DB-Library calls to get login security labels.
- DB-Library sends 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 use this method.
  - The application can call `dbsechandle` to install a user-supplied function to generate security labels. Typically, only gateway applications 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 reads 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 following example. 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,
  nbuflen, valuebuf, vbuflen, 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>`.
- *Return values for security label handlers* lists the return values that are legal for a security label handler. A security label handler must return one of these values.

Label handler return value	Indicates
DBMORELABEL	The label handler has set the name and value of a login security label. 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. DB-Library should not call the label handler again.
DBERRLABEL	A label handler error has occurred. DB-Library should abort the connection attempt.

2.161 dbsendpassthru

Send a TDS packet to a server.

Syntax

```
RETCODE dbsendpassthru(dbproc, send_bufp)

 DBPROCESS      *dbproc;
 DBVOIDPTR        send_bufp;
```

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.

send_bufp
: 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`.

Returns

DB_PASSTHRU_MORE, DB_PASSTHRU_EOM, or FAIL.

Usage

- dbsendpassthru sends a TDS (Tabular Data Stream) packet to a server.
- 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.
- 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.
- dbsendpassthru sends a packet of bytes from the buffer to which <send_bufp> points. Most commonly, <send_bufp> is *<recv_bufp> as returned by dbrecvpassthru. <send_bufp> may also be the address of a user-allocated buffer containing the packet to be sent.
- 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.
- 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.
- 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_recvpassthru() is the Open
** Server counterpart required to complete this
** passthru operation.
*/
DBPROCESS *dbproc;
SRV_PROC *srvproc;
int ret;
BYTE *packet;
while(1)
{
ret = srv_recvpassthru(srvproc, &packet,
(int *)NULL);
if( ret == SRV_S_PASSTHRU_FAIL )
{
fprintf(stderr, "ERROR - \
srv_recvpassthru 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;
}
```

Related Information

dbgetpacket [page 196]
DBSETLPACKET [page 377]
dbrecvpassthru [page 297]

2.162 dbservcharset

Get the name of the server character set.

Syntax

```
char *dbservcharset(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/C uses to manage communications and data between the front end and the server.

Returns

A pointer to the null-terminated name of the server’s character set, or NULL in case of error.

Usage

- `dbservcharset` returns the name of the server's character set.
- 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 performs all conversions to and from its own character set when communicating with the client.
- An application can inform the server what character set it is using `DBSETLCHARSET`.
- To determine if the server is performing character set translations, an application can call `dbcharsetconv`.
- To get the name of the client character set, an application can call `dbgetcharset`.

Related Information

dbcharsetconv [page 105]
dbgetcharset [page 188]
DBSETLCHARSET [page 367]

2.163 dbsetavail

Marks a DBPROCESS as being available for general use.

Syntax

```
void dbsetavail(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.

Returns

None.

Usage

This routine marks the DBPROCESS as being available for general use. Any subsequent calls to `DBISAVAIL` returns "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.

Related Information

DBISAVAIL [page 206]

2.164 dbsetbusy

Call a user-supplied function when DB-Library is reading from the server.

Syntax

```
void dbsetbusy(dbproc, busyfunc)

DBPROCESS      *dbproc;
int                       *(*busyfunc)())();
```

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.

busyfunc

The user-supplied function that DB-Library calls whenever it accesses the server. DB-Library calls `<busyfunc>`() with a single parameter—a pointer to the DBPROCESS from the `dbsetbusy` call.

<busyfunc>() returns a pointer to a function that returns an integer.

Returns

None.

Usage

- This routine associates a user-supplied function with the specified <dbproc>. The user-supplied function is 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 causes 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>() is 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>() is called multiple times.
- Here is an example of defining and installing <busyfunc>() and <idlefunc>():

  > i 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");
dbsqlexec(dbproc);
/*
** DB-Library calls busyfunc() for the first time
** during dbsqlexec(). 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);
}
```

Related Information

dbsetidle [page 359]

2.165 dbsetconnect

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 SAP Open Server do not currently use net_name; it is a placeholder should SAP 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. SAP 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.

Returns

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*.

Related Information

dbopen [page 274]

2.166 dbsetdefcharset

Set the default character set for an application.

Syntax

```
RETCODE dbsetdefcharset(charset)

char         *charset;
```

Parameters

charset
: The name of the character set to use. `<charset>` must be a null-terminated character string.

Returns

SUCCEED or FAIL.

Usage

- dbsetdefcharset sets an application's default character set.
- 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.
- If an application does not call dbsetdefcharset, its default character set is the character set of the first DBPROCESS connection opened, or iso_1 if no DBPROCESS is open.
- If an application plans to call both dbsetdefcharset and dbsetdeflang, it must call dbsetdefcharset first.

Related Information

dbsetdeflang [page 358]
dbsetdefcharset [page 357]
dblogin [page 213]
dbopen [page 274]

2.167 dbsetdeflang

Set the default language name for an application.

Syntax

```
RETCODE dbsetdeflang(language)
char          *language;
```

Parameters

language

The name of the national language to use. <language> must be a null-terminated character string.

Returns

SUCCEED or FAIL.

Usage

- dbsetdeflang sets an application's default national language.
- 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.
- 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.

Related Information

DBSETLNATLANG [page 371]

2.168 dbsetidle

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.

Returns

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>`().

Related Information

dbsetbusy [page 353]
dbsetbusy [page 353]

2.169 dbsetifile

Specify the name and location of the Sybase interfaces file.

Syntax

```
void dbsetifile(filename)

char          *filename;
```

Parameters

filename

The name of the interfaces file that gets searched during every subsequent call to `dbopen`. If this parameter is NULL, DB-Library will revert to the default file name.

Returns

None.

Usage

- This routine lets the application specify the name and location of the interfaces file that will be searched during every subsequent call to `dbopen`. The interfaces file contains the name and network address of every server available on the network.
- If `dbsetifile` has not been called, a call to `dbopen` initiates the following default behavior: DB-Library attempts to use a file named `interfaces` 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 `interfaces` in the home directory of the user named “sybase.”
- See the *Open Client and Open Server Configuration Guide*.

i 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.

Related Information

dbopen [page 274]

2.170 dbsetinterrupt

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.

Return values for the hndlintr() function lists the legal return values of hndlintr:

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.

Returns

None.

Related Information

Usage for dbsetinterrupt [page 363]

dbcancel [page 101]

dbgetuserdata [page 200]

dbsetuserdata [page 390]

dbsetbusy [page 353]

dbsetidle [page 359]

2.170.1 Usage for dbsetinterrupt

Review how to use `dbsetinterrupt`.

- 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).
    - 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.
- 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:

  i 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;
}
}
```

2.171 DBSETLAPP

Set the application name in the LOGINREC structure.

Syntax

```
RETCODE DBSETLAPP(loginrec, application)

LOGINREC     *loginrec;
char                 *application;
```

Parameters

loginrec

A pointer to a LOGINREC structure, which is passed as an argument to `dbopen`. You can allocate a LOGINREC structure by calling `dblogin`.

application

The application name that is 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.

Returns

SUCCEED or FAIL.

Usage

- This macro sets the application field in the LOGINREC structure. For it to have any effect, it must be called before `dbopen`.
- It is not necessary to call this routine. By default, the application name is a null value.
- The server uses the application name in its `sysprocesses` table to help identify your process. If you set the application name, you see it if you query the `sysprocesses` table in the `master` database.

Related Information

dblogin [page 213]
dbopen [page 274]
DBSETLHOST [page 369]
DBSETLPWD [page 378]
DBSETLUSER [page 380]

2.172 DBSETLCHARSET

Set the character set in the LOGINREC structure.

Syntax

```
RETCODE DBSETLCHARSET(loginrec, char_set)

LOGINREC    *loginrec;
DBCHAR        *char_set;
```

Parameters

loginrec

A pointer to a LOGINREC structure to be passed as an argument to `dbopen`. LOGINREC structures are obtained using `dblogin`.

char_set

The name of the character set the client 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.

Returns

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 performs 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.

Related Information

dbgetcharset [page 188]
dblogin [page 213]
dbopen [page 274]

2.173 DBSETLENCRYPT

Specify whether or not network password encryption is to be used when logging onto SAP Adaptive Server Enterprise.

Syntax

```
RETCODE DBSETLENCRYPT(loginrec, enable)

LOGINREC      *loginrec;
DBBOOL         enable;
```

Parameters

loginrec
: A pointer to a LOGINREC structure, which is 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.

Returns

SUCCEED or FAIL.

Usage

- 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.
- Network password encryption provides a protected mechanism for authenticating a user's identity.
- If an application specifies that network password encryption is to be used, then when the application attempts to open a connection:
    - No password is sent with the initial connection request. At this time, the client indicates to the server that encryption is desired.
    - The server replies to the connection request with an encryption key.
    - 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.
    - The server uses the key to decrypt the encrypted passwords and either accepts or rejects the login attempt.
- If password encryption is not specified, then when an application attempts to open a connection:
    - A password is included with the connection request.
    - The server either accepts or rejects the login attempt.

Related Information

dbsechandle [page 345]

2.174 DBSETLHOST

Set the host name in the LOGINREC structure.

Syntax

```
RETCODE DBSETLHOST(loginrec, hostname)

LOGINREC    *loginrec;
char                *hostname;
```

Parameters

loginrec

A pointer to a LOGINREC structure, which is passed as an argument to `dbopen`. You can allocate a LOGINREC structure by calling `dblogin`.

hostname

The host name that is 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.

Returns

SUCCEED or FAIL.

Usage

- This macro sets the host name in the LOGINREC structure. For it to have any effect, it must be called before `dbopen`.
- The host name shows up in the `sysprocesses` table in the `master` database.
- It is not necessary to call this routine. If it is not called, DB-Library is set the default value for the host name. This default value is a version of the host machine's name provided by the operating system.

Related Information

dblogin [page 213]
dbopen [page 274]
DBSETLAPP [page 365]
DBSETLPWD [page 378]
DBSETLUSER [page 380]

2.175 DBSETLMUTUALAUTH

Enables or disables mutual authentication of the connection's security mechanism.

Syntax

```
RETCODE DBSETLMUTUALAUTH(loginrec, enable)
LOGINREC              *loginrec;
DBBOOL                  *enable;
```

Parameters

loginrec
: A pointer to a LOGINREC structure, which is 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 enable mutual authentication.

Returns

SUCCEED or FAIL.

Usage

- For DBSETLMUTUALAUTH to take effect, it must be called before `dbopen()` and DBSETLNETWORKAUTH must be enabled.
- If DBSETLMUTUALAUTH is not called, mutual authentication is disabled by default.

Related Information

dblogin [page 213]
DBSETLNETWORKAUTH [page 372]
DBSETLSERVERPRINCIPAL [page 379]

2.176 DBSETLNATLANG

Set the national language name in the LOGINREC structure.

Syntax

```
RETCODE DBSETLNATLANG(loginrec, language)

LOGINREC    *loginrec;
char                *language;
```

Parameters

loginrec

A pointer to a LOGINREC structure to be passed as an argument to `dbopen`. LOGINREC structures are obtained using `dblogin`.

language

The name of the national language to use. `<language>` must be a null-terminated character string.

Returns

SUCCEED or FAIL.

Usage

- This macro sets the user language in the LOGINREC structure. If you wish to set a particular user language, call `DBSETLNATLANG` before `dbopen`.
- Call `DBSETLNATLANG` only if you do not wish to use the server's default national language.

Related Information

dblogin [page 213]
dbopen [page 274]
dbsetdeflang [page 358]

2.177 DBSETLNETWORKAUTH

Enables or disables network-based authentication.

Syntax

```
RETCODE DBSETLNETWORKAUTH(loginrec, enable)
LOGINREC                *loginrec;
DBBOOL                    *enable;
```

Parameters

loginrec

A pointer to a LOGINREC structure, is 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 enable network authentication.

Returns

SUCCEED or FAIL.

Usage

If DBSETLNETWORKAUTH is not called, network authentication is disabled by default.

Related Information

dblogin [page 213]
DBSETLMUTUALAUTH [page 370]
DBSETLSERVERPRINCIPAL [page 379]

2.178 dbsetloginfo

Transfer TDS login information from a DBLOGINFO structure to a LOGINREC structure.

Syntax

```
RETCODE dbsetloginfo(loginrec, loginfo)

LOGINREC        *login;
DBLOGINFO      *loginfo;
```

Parameters

login

A pointer to a LOGINREC structure. This pointer is passed as an argument to `dbopen`. You can allocate a LOGINREC structure by calling `dblogin`.

loginfo

A pointer to a DBLOGINFO structure that contains login parameter information.

Returns

SUCCEED or FAIL.

Usage

- `dbsetloginfo` transfers TDS login information from a DBLOGINFO structure to a LOGINREC structure. After the information is transferred, `dbsetloginfo` frees the DBLOGINFO structure.
- An application needs to call `dbsetloginfo` only if (1) it is an Open Server gateway application and (2) it is using TDS passthrough.
- TDS (Tabular Data Stream) 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 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.
- `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 DBLOGINFO structure and fills it with TDS information from a client SRV_PROC.
    - `dbsetloginfo` transfers the TDS information retrieved by `srv_getloginfo` from the DBLOGINFO structure to a DB-Library/C LOGINREC structure, and then frees the DBLOGINFO 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 DBLOGINFO structure.
    - `srv_setloginfo` sends the remote server's response, retrieved by `dbgetloginfo`, to the client, and then frees 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;
{
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 informationfrom
** 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);
}
```

Related Information

dbgetloginfo [page 189]
dbrecvpassthru [page 297]
dbsendpassthru [page 349]

2.179 dbsetlogintime

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

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.

Returns

SUCCEED or FAIL.

Usage

- This routine sets the length of time in seconds that DB-Library waits for a login response after calling `dbopen`. The default timeout value is 60 seconds.
- 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 signals 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 does not respond. Because "no response" is not considered to be an error, the network does not signal the client that an error has occurred. However, if `dbsetlogintime` has been called to set a timeout period, a timeout error occurs when the client fails to receive a response within the set period.

Related Information

dberrhandle [page 172]
dbsettime [page 389]

2.180 DBSETLPACKET

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 sets the actual packet size to a value less than or equal to this requested size.

Returns

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 uses 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` has no effect on the packet sizes of DBPROCESSes already allocated using `dbopen`.

Related Information

dblogin [page 213]
dbopen [page 274]
dbgetpacket [page 196]

2.181 DBSETLPWD

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 is passed as an argument to `dbopen`. You can allocate a LOGINREC structure by calling `dblogin`.

password
: The password that is 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.

Returns

SUCCEED or FAIL.

Usage

- This macro sets the user server password in the LOGINREC structure. For it to have any effect, it must be called before `dbopen`.
- 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.
- DB-Library does not automatically blank out the password in `<loginrec>` after a call to `dbopen`. Therefore, if you want to minimize the risk of having a readable password in your DB-Library program, you should set `<password>` to something else after you call `dblogin`.

Related Information

dblogin [page 213]
dbopen [page 274]
DBSETLAPP [page 365]
DBSETLHOST [page 369]
DBSETLUSER [page 380]

2.182 DBSETLSERVERPRINCIPAL

Sets the server's principal name in the LOGINREC structure, if required.

Syntax

```
DBSETLSERVERPRINCIPAL(loginrec, name)
LOGINREC            *loginrec;
char                        *name;
```

Parameters

loginrec

A pointer to a LOGINREC structure, which is passed as an argument to `dbopen`. You can allocate a LOGINREC structure by calling `dblogin`.

name

The server's principal name. The maximum length of the string, not including the null terminator, is 255 characters.

Returns

SUCCEED or FAIL.

Usage

- For DBSETLSERVERPRINCIPAL to take effect, it must be called before `dbopen()` and DBSETLNETWORKAUTH must be enabled.
- If DBSETLSERVERPRINCIPAL is not called, the server name is set as the principal name.

Related Information

dblogin [page 213]
DBSETLMUTUALAUTH [page 370]
DBSETLNETWORKAUTH [page 372]

2.183 DBSETLUSER

Set the user name in the LOGINREC structure.

Syntax

```
RETCODE DBSETLUSER(loginrec, username)

LOGINREC     *loginrec;
char                *username;
```

Parameters

loginrec

A pointer to a LOGINREC structure, which is passed as an argument to `dbopen`. You can allocate a LOGINREC structure by calling `dblogin`.

username

The user name that is 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 uses `<username>` to determine who is attempting the connection. The server usernames are defined in the `syslogins` table in the `master` database.

Returns

SUCCEED or FAIL.

Usage

- This macro sets the user name in the LOGINREC structure. For it to have any effect, it must be called before `dbopen`.
- In most environments, this macro is optional. If it is not called, DB-Library sets the default value for the user name.

i Note

On *UNIX*: the user name defaults to the UNIX login name.

On *MPE/XL*: The user name defaults to the value of the system environment variable HPUSER.

Related Information

dblogin [page 213]
dbopen [page 274]
DBSETLHOST [page 369]
DBSETLPWD [page 378]
DBSETLAPP [page 365]

2.184 dbsetmaxprocs

Set the maximum number of simultaneously open DBPROCESS structures.

Syntax

```
RETCODE dbsetmaxprocs(maxprocs)

int           maxprocs;
```

Parameters

maxprocs	The new limit on simultaneously open DBPROCESS structures for this particular program.

Returns

SUCCEED or FAIL.

Usage

- 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`.
- The program may find out what the current limit is by calling `dbgetmaxprocs`.

Related Information

dbgetmaxprocs [page 192]
dbopen [page 274]

2.185 dbsetnull

Define substitution values to be used when binding null values.

Syntax

```
RETCODE dbsetnull(dbproc, bindtype, bindlen, bindval)

DBPROCESS    *dbproc;
int                        bindtype;
int                        bindlen;
BYTE                 *bindval;
```

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.

bindtype

A symbolic value specifying the type of variable binding to which the substitute value applies. (See the reference page for dbbind.)

bindlen

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.

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.

Returns

SUCCEED or FAIL.

`dbsetnull` returns FAIL if you give it an unknown `<bindtype>`. It also fails 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. *Default null substitution values* lists the default 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 remains 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 sets the indicator value to indicate null data values or conversion errors.

Related Information

dbbind [page 88]
dbaltbind [page 68]
dbbind [page 88]
dbconvert [page 124]
dbnullbind [page 266]
Types [page 470]

2.186 dbsetopt

Set a server or DB-Library option.

Syntax

```
RETCODE dbsetopt(dbproc, option, char_param,
               int_param)

DBPROCESS     *dbproc;
int               option;
char          *char_param;
int               int_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 is set for all active DBPROCESS structures.

option

The option that is to be turned on.

char_param

Certain options take parameters. For example, the DBOFFSET option takes as its parameter the construct for which offsets are to be returned:

```
dbsetopt(dbproc, DBOFFSET, "compute", -1)
```

The DBBUFFER option takes as its parameter the number of rows to be buffered:

```
dbsetopt(dbproc, DBBUFFER, "500", -1)
```

`<char_param>` 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 is discovered the next time a command buffer is sent to the server. The `dbsqlexec` or `dbsqlsend` call fails, and DB-Library invokes the user-installed message handler. If an invalid parameter is specified for one of the DB-Library options (DBBUFFER or DBTEXTLIMIT), the `dbsetopt` call itself fails.

If the option takes no parameters, `<char_param>` must be NULL.

int_param

Some options require an additional parameter, `<int_param>`, which is the length of the character string passed as `<char_param>`. Currently, only DBPRCOLSEP, DBPRLINESEP, and DBPRPAD require this parameter.

If `<int_param>` is not required, pass it as -1.

Returns

SUCCEED or FAIL.

`dbsetopt` fails if `<char_param>` is invalid for one of the DB-Library options. However, an invalid `<char_param>` for a server option does 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 `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.
- `dbsetopt` does not immediately set the option. The option is set the next time a command buffer is sent to the server (by invoking `dbsqlexec` or `dbsqlsend`).

Related Information

Options [page 466]

Options [page 466]
dbclropt [page 108]
dbisopt [page 207]
Options [page 466]
Options [page 466]

2.187 dbsetrow

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.

Returns

MORE_ROWS, NO_MORE_ROWS, or FAIL.

`dbsetrow` returns:

- MORE_ROWS if it found `<row>` in the row buffer, or
- 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.

Usage

- dbsetrow sets a buffered row to “current.” After dbsetrow is called, the application’s next call to dbnextrow reads 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 causes 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 with dbsetrow. Thus, the call:

  ```
  dbsetrow(dbproc, DBFIRSTROW(dbproc))
  ```

  sets the current row so that the next call to dbnextrow reads the first row in the buffer.

Related Information

dbclrbuf [page 107]
DBCURROW [page 136]
DBFIRSTROW [page 180]
dbgetrow [page 197]
DBLASTROW [page 209]
dbnextrow [page 260]
Options [page 466]

2.188 dbsettime

Set the number of seconds that DB-Library waits for a server response to a SQL command.

Syntax

```
RETCODE dbsettime(seconds)

int            seconds;
```

Parameters

seconds

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.

Returns

SUCCEED or FAIL.

Usage

- This routine sets the length of time in seconds that DB-Library waits for a server response during calls to `dbsqlexec`, `dbsqlok`, `dbresults`, and `dbnextrow`. The default timeout value is 0, which represents an infinite timeout period.
- `dbsettime` can be called at any time during the application—before or after a call to `dbopen`. It takes effect immediately upon being called.
- To set a timeout value for calls to `dbopen`, use `dbsetlogintime`.
- Note that, after sending a query to the server, `dbsqlexec` 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 `dbsqlsend`, followed by `dbsqlok`.
- The program can call `DBGETTIME` to learn the current timeout value.
- A timeout generates the DB-Library error “SYBETIME.”

Related Information

dberrhandle [page 172]
DBGETTIME [page 199]
dbsetlogintime [page 375]
dbsqlexec [page 402]
dbsqlok [page 404]
dbsqlsend [page 409]

2.189 dbsetuserdata

Use a DBPROCESS structure to save a pointer to user-allocated data.

Syntax

```
void dbsetuserdata(dbproc, ptr)

DBPROCESS     *dbproc;
BYTE                    *ptr;
```

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.

ptr

A generic BYTE pointer to the user's private data space.

Returns

None.

Usage

- This routine saves, in a DBPROCESS structure, a pointer to user-allocated data. The application can access the data later with the `dbgetuserdata` routine.
- `dbsetuserdata` 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 following example. This routine is particularly useful when the application has multiple DBPROCESS structures.
- The application must allocate the data that `<ptr>` points to. DB-Library never manipulates this data; it merely saves the pointer to it for later use by the application.
- Here is an example of using this routine to handle deadlock, a situation which occurs in high-volume applications. See the *SAP Adaptive Server Enterprise System Administration Guide*. This program fragment sends updates to the server. It reruns the transaction when its message handler detects deadlock.

```
                ...
/*
** 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 ......");
dbsqlexec(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 ......");
```

```
dbsqlexec(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");
dbsqlexec(dbproc);
dbresults(dbproc);
...
/*
** SERVERMSGS
** 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 */
}
```

Related Information

dbgetuserdata [page 200]

2.190 dbsetversion

Specify a DB-Library version level.

Syntax

```
RETCODE dbsetversion(version)
```

```
DBINT          version;
```

Parameters

version

The version of DB-Library behavior that the application expects. The symbolic values that are legal for `<version>` are:

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. This is the default version of DB-Library.
DBVERSION_100	10.0 behavior	`numeric` and `decimal` datatypes.

Returns

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.

i 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.

Related Information

dbinit [page 203]

2.191 dbspid

Get the server process ID for the specified DBPROCESS.

Syntax

```
int dbspid(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.

Returns

`<dbproc>`'s server process ID.

Usage

- `dbspid` yields the server process ID of the specified DBPROCESS. The process ID appears in the server's `sysprocesses` table.
- You can use the server process ID to make queries against the `sysprocesses` table.

Related Information

dbopen [page 274]

2.192 dbspr1row

Place one row of server query results into a buffer.

Syntax

```
RETCODE dbspr1row(dbproc, buffer, buf_len)

DBPROCESS     *dbproc;
char                    *buffer;
DBINT                  buf_len;
```

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.

buffer

A pointer to a character buffer to contain the `dbspr1row` results.

buf_len

The length of `<buffer>`, including its null terminator.

Returns

SUCCEED or FAIL.

i Note

If an error occurs, the contents of *`<buffer>` are undefined.

Usage

- `dbspr1row` fills a programmer-supplied buffer with a null-terminated character string containing one server query results row.
- `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` 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 is 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 [page 466] for more details on the DBPRPAD option.
- You can specify the column separator string using the DB-Library option DBPRCOLSEP. The column separator is added to the end of each converted column's data except the last. The default separator is an ASCII 0x20 (space).
- 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).
- 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 );
```

Related Information

Options [page 466]
Options [page 466]
Options [page 466]
dbclropt [page 108]
dbisopt [page 207]
dbprhead [page 283]
dbprrow [page 284]
dbspr1rowlen [page 397]
dbsprhead [page 398]
dbsprline [page 400]
Options [page 466]

2.193 dbspr1rowlen

Determine how large a buffer to allocate to hold the results returned by `dbsprhead`, `dbsprline`, and `dbspr1row`.

Syntax

```
DBINT dbspr1rowlen(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.

Returns

The size of the buffer, in bytes, required by `dbsprhead`, `dbsprline`, and `dbspr1row` on success; a negative integer on error.

Usage

- `dbspr1rowlen` determines the size of the buffer (in bytes) required by `dbsprhead, dbsprline`, and `dbspr1row`, including the null terminator.
- `dbspr1rowlen` is useful when printing data for debugging and when scrolling data displays.
- To make the best use of `dbspr1rowlen`, application programs should call it once for every successful call to `dbresults`.
- The following code fragment illustrates the use of `dbspr1rowlen`:

  ```
  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", dbspr1rowlen(dbproc));
  ```

Related Information

dbprhead [page 283]
dbprrow [page 284]
dbspr1row [page 395]
dbsprhead [page 398]
dbsprline [page 400]
Options [page 466]

2.194 dbsprhead

Place the server query results header into a buffer.

Syntax

```
RETCODE dbsprhead(dbproc, buffer, buf_len)

DBPROCESS     *dbproc;
char                    *buffer;
DBINT                  buf_len;
```

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.
buffer	A pointer to a character buffer to contain the query results header.
buf_len	The length of `<buffer>`, including its null terminator.

Returns

SUCCEED or FAIL.

> **i 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 is 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 [page 466] for more details on the DBPRPAD option.
- You can specify the column separator string using the DB-Library option DBPRCOLSEP. The column separator is added to the end of each column name except the last. The default separator is an ASCII 0x20 (space).
- 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* 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");
  dbsqlexec(dbproc);
  dbresults(dbproc);
  dbsprhead(dbproc, buffer, sizeof(buffer));
  printf("%s\n", buffer);
  ```

Related Information

Options [page 466]

Options [page 466]

Options [page 466]

dbprhead [page 283]

dbprrow [page 284]

dbsetopt [page 385]

dbspr1row [page 395]

dbspr1rowlen [page 397]

dbsprline [page 400]

Options [page 466]

2.195 dbsprline

Get a formatted string that contains underlining for the column names produced by `dbsprhead`.

Syntax

```
RETCODE dbsprline(dbproc, buffer, buf_len, linechar)

DBPROCESS     *dbproc;
char                    *buffer;
DBINT                  buf_len;
DBCHAR              linechar;
```

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.

buffer

A pointer to a character buffer to contain the `dbsprline` results.

buf_len

The length of `<buffer>`, including its null terminator.

linechar

The character with which to "underline" column names produced by `dbsprhead`.

Returns

SUCCEED or FAIL.

i 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 */
```

Related Information

dbprhead [page 283]
dbprrow [page 284]
dbspr1row [page 395]
dbspr1rowlen [page 397]
dbsprhead [page 398]
Options [page 466]

2.196 dbsqlexec

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.

Returns

SUCCEED or FAIL.

The most common reason for failing is a SQL syntax error. `dbsqlexec` fails 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 causes `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 does not cause `dbsqlexec` to fail. Instead, failure occurs 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 does 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` still returns 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 `dbfcmd`.
- 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 */
dbsqlexec(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
}
```

- `dbsqlexec` is equivalent to `dbsqlsend` followed by `dbsqlok`. However, after sending a query to the server, `dbsqlexec` waits until a response is received or until the timeout period has elapsed. By substituting `dbsqlsend` and `dbsqlok` for `dbsqlexec`, 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 `dbsqlok`.
- Multiple commands may exist in the command buffer when an application calls `dbsqlexec`. These commands are sent to the server as a unit and are considered to be a single command batch.

Related Information

dbsqlsend [page 409]
dbsqlok [page 404]
dbcmd [page 110]
dbfcmd [page 177]
dbnextrow [page 260]
dbresults [page 320]
dbretstatus [page 329]
dbsettime [page 389]
dbsqlok [page 404]
dbsqlsend [page 409]

2.197 dbsqlok

Wait for results from the server and verify the correctness of the instructions the server is responding to.

Syntax

```
RETCODE dbsqlok(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.

Returns

SUCCEED or FAIL.

The most common reason for failing is a SQL syntax error. `dbsqlok` fails 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, causes `dbsqlok` to fail *if* the command buffer contains only a single command. If the command buffer contains multiple commands, a runtime error does not cause `dbsqlok` to fail. Instead, failure occurs 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 `dbsqlok` to fail, in accordance with the rule given in the previous paragraph. A runtime error on a statement inside a stored procedure does not cause `dbsqlok` 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 `dbsqlok` still returns 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.

Related Information

Usage for dbsqlok [page 405]

dbcmd [page 110]

dbfcmd [page 177]

DBIORDESC [page 203]

DBIOWDESC [page 205]

dbmoretext [page 254]

dbnextrow [page 260]

dbpoll [page 279]

DBRBUF [page 291]

dbresults [page 320]

dbretstatus [page 329]

dbrpcsend [page 339]

dbsettime [page 389]

dbsqlexec [page 402]

dbsqlsend [page 409]

dbwritetext [page 443]

2.197.1 Usage for dbsqlok

Review the use of dbsqlok.

- `dbsqlok` reports the success or failure of a server command and initiates results processing for successful commands.
- A successful `dbsqlok` call must always be followed by a call to `dbresults` to process the results.
- `dbsqlok` is useful in the following situations:
  - After a `dbsqlsend` call
    `dbsqlok` must be called after a batch of Transact-SQL commands is sent to the server with `dbsqlsend`.

    - After a dbrpcsend call
      dbsqlok must be called after an RPC command is sent with dbrpcinit, dbrpcparam, and dbrpcsend.
    - After calls to dbwritetext or dbmoretext
      dbsqlok must be called after a text update command is sent to the server by a call to dbwritetext or dbmoretext.

Using dbsqlok with dbsqlsend

- dbsqlok initiates results processing after a call to dbsqlsend.
- dbsqlok 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 dbsqlok 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 dbsqlok reports success or failure and initiates results processing if successful.

  i Note

  On occasion, dbpoll may report that data is ready for dbsqlok to read when only the first bytes of the server response are present. When this occurs, dbsqlok 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 dbsqlok 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 dbsqlok() 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 dbsqlok().
```

```
*/
retcode = dbsqlok(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 dbsqlok().
*/
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';
```

Using dbsqlok with dbrpcsend

- `dbsqlok` initiates results processing after an RPC command. RPC commands are constructed and sent with `dbrpcinit`, `dbrpcparam`, and `dbrpcsend`. After `dbrpcsend`, the program must call `dbsqlok`.
- `dbpoll` can be called in a loop to poll for a server response between `dbrpcsend` and `dbsqlok`.
- The sample program `example8.c` demonstrates an RPC command.

Using dbsqlok with dbwritetext and dbmoretext

- `dbsqlok` 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, `dbsqlok` must be called.
- The reference pages for `dbwritetext` has an example.

Related Information

dbrpcinit [page 335]
dbrpcparam [page 337]
dbrpcsend [page 339]
dbwritetext [page 443]
dbmoretext [page 254]

2.198 dbsqlsend

Send a command batch to the server and do not wait for a response.

Syntax

```
RETCODE dbsqlsend(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.

Returns

SUCCEED or FAIL.

`dbsqlsend` may fail if previous results had not been processed, or if the command buffer was empty.

Usage

- 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 `dbfcmd`.
- 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.
- `dbsqlexec` is equivalent to `dbsqlsend` followed by `dbsqlok`.
- The use of `dbsqlsend` with `dbsqlok` is of particular value in UNIX applications. After sending a query to the server, `dbsqlexec` waits until a response is received or until the timeout period has elapsed. By substituting `dbsqlsend`, `dbpoll` and `dbsqlok` for `dbsqlexec`, you can sometimes provide a way for an application to respond more effectively to multiple input and output streams. See the `dbsqlok` reference page.

Related Information

dbsqlok [page 404]
dbcmd [page 110]
dbfcmd [page 177]
DBIORDESC [page 203]
DBIOWDESC [page 205]
dbnextrow [page 260]
dbpoll [page 279]
dbresults [page 320]
dbsettime [page 389]
dbsqlexec [page 402]
dbsqlok [page 404]

2.199 dbstrbuild

Build a printable string from text containing placeholders for variables.

Syntax

```
int dbstrbuild(dbproc, charbuf, bufsize,
             text [, formats [, arg] ... ])

DBPROCESS     *dbproc;
char                    *charbuf;
int                         bufsize;
char                    *text;
char                    *formats;
???                    args...;
```

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 contains 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 are converted according to the contents of the `<formats>` string. There must be one argument for each format in the `<formats>` string. The first value corresponds to the “%1!” parameter, the second the “%2!”, and so on. 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.

Returns

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 preceding messages is:

```
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 on.

Related Information

dbconvert [page 124]
dbdatename [page 161]
dbdatepart [page 165]

2.200 dbstrcmp

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.

Returns

- 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);
  ```

Related Information

dbfreesort [page 185]
dbloadsort [page 212]
dbstrsort [page 417]

2.201 dbstrcpy

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.

Returns

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);
```

Related Information

dbcmd [page 110]
dbfcmd [page 177]
dbfreebuf [page 183]
dbgetchar [page 186]
dbstrlen [page 416]

2.202 dbstrlen

Return the length, in characters, of the command buffer.

Syntax

```
int dbstrlen(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.

Returns

The length, in characters, of the command buffer.

Usage

- `dbstrlen` returns the length, in characters, of the SQL command text in the command buffer.
- 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.
- Before you copy the command buffer with `dbstrcpy`, use `dbstrlen` to make sure that the destination buffer is large enough.
- The count returned by `dbstrlen` does not include space for a null terminator.

Related Information

dbcmd [page 110]
dbfcmd [page 177]
dbfreebuf [page 183]
dbgetchar [page 186]
dbstrcpy [page 414]

2.203 dbstrsort

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.

Returns

- 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.

Following 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.

Related Information

dbfreesort [page 185]
dbloadsort [page 212]
dbstrcmp [page 412]

2.204 dbtabbrowse

Determine whether the specified table is updatable through the DB-Library browse-mode facilities.

Syntax

```
DBBOOL dbtabbrowse(dbproc, tabnum)

DBPROCESS    *dbproc;
int                         tabnum;
```

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.

tabnum

The number of the table of interest, as specified in the `select` statement's `from` clause. Table numbers start at 1.

Returns

"TRUE" or "FALSE".

Usage

- dbtabbrowse is one of the DB-Library browse-mode routines. See Browse Mode [page 44] for a detailed discussion of browse mode.
- 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.
- For a table to be considered "browsable," it must have a unique index and a timestamp column.
- The application can call dbtabbrowse anytime after dbresults.
- The sample program example7.c contains a call to dbtabbrowse.

Related Information

dbcolbrowse [page 113]
dbcolsource [page 118]
dbqual [page 288]
dbtabcount [page 420]
dbtabname [page 422]
dbtabsource [page 423]
dbtsnewlen [page 427]
dbtsnewval [page 428]
dbtsput [page 430]
Browse Mode [page 44]

2.205 dbtabcount

Return the number of tables involved in the current select query.

Syntax

```
int dbtabcount(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.

Returns

The number of tables, including server work tables, involved in the current set of row results.

`dbtabcount` returns -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 [page 44] 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`.

Related Information

dbcolbrowse [page 113]
dbcolsource [page 118]
dbqual [page 288]

dbtabbrowse [page 419]
dbtabname [page 422]
dbtabsource [page 423]
dbtsnewlen [page 427]
dbtsnewval [page 428]
dbtsput [page 430]
Browse Mode [page 44]

2.206 dbtabname

Return the name of a table based on its number.

Syntax

```
char *dbtabname(dbproc, tabnum)

DBPROCESS      *dbproc;
int                             tabnum;
```

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.

tabnum

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.

Returns

A pointer to the null-terminated name of the specified table. This pointer is NULL if the table number is out of range or if the specified table is a server work table. For a description of work tables, see dbtabcount [page 420].

Usage

- `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`). For a detailed discussion of browse mode, see Browse Mode [page 44].
- 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.
- The application can call `dbtabname` anytime after `dbresults`.
- The sample program `example7.c` contains a call to `dbtabname`.

Related Information

dbtabcount [page 420]
dbcolbrowse [page 113]
dbcolsource [page 118]
dbqual [page 288]
dbtabbrowse [page 419]
dbtabcount [page 420]
dbtabsource [page 423]
dbtsnewlen [page 427]
dbtsnewval [page 428]
dbtsput [page 430]
Browse Mode [page 44]

2.207 dbtabsource

Return the name and number of the table from which a particular result column was derived.

Syntax

```
char *dbtabsource(dbproc, colnum, tabnum)

DBPROCESS       *dbproc;
int                       colnum;
int                      *tabnum;
```

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.

colnum

The number of the result column of interest. Column numbers start at 1.

tabnum

A pointer to an integer, which is 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), `<*tabnum>` is set to -1.

Returns

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:

- The DBPROCESS is dead or not enabled. This error invokes the application's handler.
- The column number is out of range.
- The column is the result of an expression, such as `max(colname)`.

Usage

- `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`). For a detailed discussion of browse mode, see Browse Mode [page 44].
- `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.
- The application can call `dbtabsource` anytime after `dbresults`.
- The sample program `example7.c` contains a call to `dbtabsource`.

Related Information

dbcolbrowse [page 113]
dbcolsource [page 118]
dbqual [page 288]
dbtabbrowse [page 419]

dbtabcount [page 420]
dbtabname [page 422]
dbtsnewlen [page 427]
dbtsnewval [page 428]
Browse Mode [page 44]

2.208 DBTDS

Determine which version of TDS (the Tabular Data Stream protocol) is being used.

Syntax

```
int DBTDS(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.

Returns

The version of TDS used by `<dbproc>` to communicate with the server. Currently, the possible versions are:

- DBTDS_2_0
- DBTDS_3_4
- DBTDS_4_0
- DBTDS_4_2
- DBTDS_4_6
- DBTDS_4_9_5
- DBTDS_5_0

DBTDS returns a negative integer on error.

Usage

- DBTDS returns the version of TDS (Tabular Data Stream protocol) being used by `<dbproc>` to communicate with the server.

Related Information

dbversion [page 439]

2.209 dbtextsize

Returns the number of bytes of `text` or `image` data that remain to be read for the current row.

Syntax

```
DBINT dbtextsize(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.

Returns

The following table lists the return values for `dbtextsize`:

`dbtextsize` **returns**	**To indicate**
>= 0	The number of bytes that remain to be read. Zero indicates NO_MORE_ROWS.

dbtextsize returns	To indicate
-1	An error has occurred.
-2	dbtextsize has been called for RPC data.

Usage

- dbtextsize assumes that there is only one column and that this column is of datatype text or image.
- dbtextsize is useful when an application does not know how large a text or image value is.
- dbtextsize does not work with RPC text data.

Related Information

dbreadtext [page 293]

2.210 dbtsnewlen

Return the length of the new value of the <timestamp> column after a browse-mode update.

Syntax

```
int dbtsnewlen(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.

Returns

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 `tsequal` built-in function), `dbtsnewlen` returns -1.

Usage

- `dbtsnewlen` is one of the DB-Library browse-mode routines. For a detailed discussion of browse mode, see Browse Mode [page 44].
- `dbtsnewlen` provides information about 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). The `dbtsnewlen` function allows the application to save the length of the new timestamp value, possibly for use with `dbtsput`.

Related Information

dbcolbrowse [page 113]
dbcolsource [page 118]
dbqual [page 288]
dbtabbrowse [page 419]
dbtabcount [page 420]
dbtabname [page 422]
dbtabsource [page 423]
dbtsnewval [page 428]
dbtsput [page 430]
Browse Mode [page 44]

2.211 dbtsnewval

Return the new value of the `<timestamp>` column after a browse-mode update.

Syntax

```
DBBINARY *dbtsnewval(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.

Returns

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 is NULL.

Usage

- `dbtsnewval` is one of the DB-Library browse-mode routines. For a detailed discussion of browse mode, see Browse Mode [page 44].
- `dbtsnewval` provides information about 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). This routine allows the application to save the new timestamp value, possibly for use with `dbtsput`.

Related Information

dbtabbrowse [page 419]
dbtabsource [page 423]
dbqual [page 288]
dbtabbrowse [page 419]
dbtabcount [page 420]
dbtabname [page 422]
dbtabsource [page 423]
dbtsnewlen [page 427]
dbtsput [page 430]
Browse Mode [page 44]

2.212 dbtsput

Put the new value of the <timestamp> column into the given table's current row in the DBPROCESS.

Syntax

```
RETCODE dbtsput(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 is 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 is used to identify the table.

Returns

SUCCEED or FAIL.

The following situations cause this routine to return FAIL:

- The application tries to update the timestamp of a non-existent row.
- The application tries to update the timestamp using NULL as the timestamp value (`<newts>`).
- The specified table is non-browsable.

Usage

- `dbtsput` is one of the DB-Library browse-mode routines. For a detailed discussion of browse mode, see Browse Mode [page 44].
- `dbtsput` 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.
  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.

Related Information

dbcolbrowse [page 113]
dbcolsource [page 118]
dbqual [page 288]
dbtabbrowse [page 419]
dbtabcount [page 420]
dbtabname [page 422]
dbtabsource [page 423]
dbtsnewlen [page 427]
dbtsnewval [page 428]
Browse Mode [page 44]

2.213 dbtxptr

Return the value of the text pointer for a column in the current row.

Syntax

```
DBBINARY *dbtxptr(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 select list column of interest. Column numbers start at 1.

Returns

A DBBINARY pointer to the text pointer for the column of interest. This pointer may be NULL.

Usage

- 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.
- It is important that all the rows of the specified `text` or `image` column have valid text pointers. A `text` or `image` column row has a valid text pointer if it contains data. However, if the `text` or `image` column row contains a null value, its text pointer is valid only if the null value was explicitly entered with the `update` statement.
  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:

  ```
  update textnull
  ```

  ```
  set x = null
  ```

  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:

  ```
  insert textnull (key)
  ```

  ```
  values (2)
  ```

  When dealing with a null `text` or `image` value, be sure to use `update` to get a valid text pointer.

- 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`. If no `select` is performed prior to the call to `dbtxptr`, the call results in a DB-Library error message.
- For an example that uses `dbtxptr`, see dbwritetext [page 443].

Related Information

dbwritetext [page 443]
dbtxtimestamp [page 433]
dbwritetext [page 443]

2.214 dbtxtimestamp

Return the value of the text timestamp for a column in the current row.

Syntax

```
DBBINARY *dbtxtimestamp(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 select list column of interest. Column numbers start at 1.

Returns

A DBBINARY pointer to the text timestamp for the column of interest. This pointer may be NULL.

Usage

- 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 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.
- The length of a non-NULL text timestamp is always DBTXTSLEN (currently defined as 8 bytes).
- 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`.
  If no `select` is performed prior to the call to `dbtxtimestamp`, the call results in a DB-Library error message.
- For an example that uses `dbtxtimestamp`, see dbwritetext [page 443].

Related Information

dbwritetext [page 443]
dbtxptr [page 431]
dbwritetext [page 443]

2.215 dbtxtsnewval

Return the new value of a text timestamp after a call to `dbwritetext`.

Syntax

```
DBBINARY *dbtxtsnewval(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.

Returns

A pointer to the new text timestamp value for the SYBTEXT or SYBIMAGE value modified by a `dbwritetext` operation. This pointer may be NULL.

Usage

- 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 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`.
- After each successful `dbwritetext` operation (which may include a number of calls to `dbmoretext`), the server sends the updated text timestamp value back to DB-Library. `dbtxtsnewval` provides a way for the application to get this new timestamp value.
- The application can use `dbtxtsnewval` in two ways. First, the return from `dbtxtsnewval` can be used as the `<timestamp>` parameter of a `dbwritetext` call. Second, `dbtxtsnewval` and `dbtxtsput` can be used together 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.

Related Information

dbmoretext [page 254]
dbtxtimestamp [page 433]
dbtxtsput [page 435]
dbwritetext [page 443]

2.216 dbtxtsput

Put the new value of a text timestamp into the specified column of the current row in the DBPROCESS.

Syntax

```
RETCODE dbtxtsput(dbproc, newtxts, colnum)

DBPROCESS    *dbproc;
```

```
DBBINARY          *newtxts;
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.

newtxts

A pointer to the new text timestamp value. It is returned by `dbtxtsnewval`.

colnum

The number of the select list column of interest. Column numbers start at 1.

Returns

SUCCEED or FAIL.

Usage

- 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`.
- 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.

Related Information

dbmoretext [page 254]
dbtxtimestamp [page 433]
dbtxtsnewval [page 434]

dbwritetext [page 443]

2.217 dbuse

Use a particular database.

Syntax

```
RETCODE dbuse(dbproc, dbname)

DBPROCESS   *dbproc;
char                  *dbname;
```

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 to use.

Returns

SUCCEED or FAIL.

Usage

- This routine issues a Transact-SQL `use` command for the specified database for a particular DBPROCESS. It sets up the command and calls `dbsqlexec` and `dbresults`.
- If the `use` command fails because the requested database has not yet completed a recovery process, `dbuse` will continue to send `use` commands at one second intervals until it either succeeds or encounters some other error.
- The routine uses the `<dbproc>` provided by the caller. It also uses the command buffer of that dbproc. `dbuse` overwrites any existing commands in the buffer and clears the buffer when it is finished.

Related Information

dbchange [page 104]
dbname [page 259]

2.218 dbvarylen

Determine whether the specified regular result column's data can vary in length.

Syntax

```
DBBOOL dbvarylen(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 regular result column of interest. The first column is number 1.

Returns

"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.

Usage

- This routine indicates whether a particular regular (that is, non-compute) result column's data can vary in length. It returns "TRUE" if the result column is derived from a database column of type `varchar`, `varbinary`, `text`, `image`, `boundary`, or `sensitivity`. It returns "TRUE" if the source database column is defined as NULL, meaning that it may contain a null value.

- 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.
- You can use `dbcoltype` to determine a column's datatype. See Types [page 470] for a list of datatypes.

Related Information

dbcoltype [page 119]
Types [page 470]
dbcollen [page 115]
dbcolname [page 116]
dbcoltype [page 119]
dbdata [page 151]
dbdatlen [page 168]
dbnumcols [page 269]
dbprtype [page 286]

2.219 dbversion

Determine which version of DB-Library is in use.

Syntax

```
char *dbversion()
```

Parameters

None.

Returns

A pointer to a character string containing the version of DB-Library in use.

Usage

`dbversion` returns a pointer to a character string that contains the version number for the DB-Library that is currently in use.

Related Information

DBTDS [page 425]

2.220 dbwillconvert

Determine whether a specific datatype conversion is available within DB-Library.

Syntax

```
DBBOOL dbwillconvert(srctype, desttype)

int       srctype;
int      desttype;
```

Parameters

srctype

The datatype of the data that is to be converted. This parameter can be any of the server datatypes, as listed in *Server and DB-Library Datatypes*.

desttype

The datatype that the source data is to be converted into. This parameter can be any of the server datatypes, as listed in *Server and DB-Library Datatypes*.

Returns

“TRUE” if the datatype conversion is supported, “FALSE” if the conversion is not supported.

Usage

- 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.
- `dbconvert` can convert data stored in any of the server datatypes (although, of course, not all conversions are legal). *Server and DB-Library Datatypes* lists the Server and DB-Library datatypes.

  Server and DB-Library Datatypes

Server type	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

- *Server and DB-Library Datatypes* 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.

Related Information

dbconvert [page 124]
dbconvert_ps [page 128]
dbaltbind [page 68]
dbbind [page 88]
dbconvert [page 124]
dbconvert_ps [page 128]
Types [page 470]
dbconvert [page 124]
dbconvert_ps [page 128]

2.221 dbwritepage

Write a page of binary data to the server.

> ⚠ Caution
>
> 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, SAP Adaptive Server Enterprise database pages are 2048 bytes long.

buf

A pointer to a buffer that holds the data to be written.

Returns

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.

Related Information

dbmsghandle [page 255]
dbreadpage [page 292]

2.222 dbwritetext

Send a `text` or `image` value to the server.

Syntax

```
RETCODE dbwritetext(dbproc, objname, textptr,
                  textptrlen, timestamp, log,
                  size, text)

DBPROCESS    *dbproc;
char                   *objname;
DBBINARY        *textptr;
DBTINYINT         textptrlen;
DBBINARY        *timestamp;
DBBOOL             log;
```

```
DBINT                    size;
BYTE                    *text;
```

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.

objname

The database table and column name that is separated by a period.

textptr

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.

textptrlen

This parameter is included for future compatibility. For now, its value must be the defined constant DBTXPLEN.

timestamp

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.

log

A boolean value specifying whether this `dbwritetext` operation should be recorded in the transaction log.

size

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, `<size>` must not exceed the actual size of the value.

text

The address of a buffer containing the `text` or `image` value to be written. If this pointer is NULL, the application must call `dbmoretext` one or more times, until all `<size>` bytes of data have been sent to the server.

Returns

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 dbtxtimestamp.
  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 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 *SAP 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 *SAP Adaptive Server Enterprise System Administration Guide*, the create database reference page, and the sp_logdevice reference page in the *SAP 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 *SAP 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 long data buffers.
- Sending an entire text or image value requires a non-NULL <text> parameter. Then, dbwritetext executes 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");
dbsqlexec(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` returns control to the application immediately after notifying the server that a text transfer is about to begin. The actual text is 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");
dbsqlexec(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);
dbsqlok(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);
dbsqlok(u_dbproc);
dbresults(u_dbproc);
dbexit();
```

Note the required calls to `dbsqlok` 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 calls `dbsetopt` to make `<@@textsize>` larger.
- The DB-Library/C option DBTEXTLIMIT limits the size of `text` or `image` values that DB-Library/C reads.

Related Information

dbtxptr [page 431]

dbmoretext [page 254]

dbmoretext [page 254]

dbtxptr [page 431]

dbtxtimestamp [page 433]

dbwritetext [page 443]

dbtxtsput [page 435]

2.223 dbxlate

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. The possible values for `<status>` are:

Values for Status

Value of `<status>`	To indicate
DBXLATE_XOF	The translated string overflowed `<dest>`.
DBXLATE_XOK	The translation succeeded.
DBXLATE_XPAT	The last bytes of `<src>` are the beginning of a pattern for which there is a translation. These bytes were not translated.

Returns

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: \
\n\t%s\n\n", destbuf);
dbfree_xlate((DBPROCESS *)NULL, xlt_tosrv,
xlt_todisp);
```

Related Information

dbload_xlate [page 210]
dbfree_xlate [page 181]

2.224 Errors

The complete collection of DB-Library errors and error severities.

Syntax

```
#include <sybfront.h>

#include <sybdb.h>

#include <syberror.h>
```

Related Information

Usage for Errors [page 450]
DBDEAD [page 171]
dberrhandle [page 172]

2.224.1 Usage for Errors

Review the information about the usage for errors.

- This is the complete list of possible DB-Library errors and error severities.
- The error values are listed alphabetically in . 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.
- Provides a list of all possible error severities, with their numerical equivalents and an explanation of the type of error.
- When an error or informational event occurs, these numbers are passed to the application's current error handler (if any). An application calls `dberrhandle` to install an error handler.
- Error values are defined in the header file `sybdb.h`. Error severity values are defined in the header file `syberror.h`. Your program must include `syberror.h` only if it refers to the symbolic error severities.

Related Information

Error Severities [page 451]
Errors [page 452]

2.224.1.1 Error Severities

Review the error severities.

The meanings for each symbolic error severity value.

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 SAP Technical Support.

2.224.1.2 Errors

Review the list of all the DB-Library errors.

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 `dblib.loc` localization file.
SYBEASEC	EXPROGRAM	Attempt to send an empty command buffer to the server.
SYBEASNL	EXPROGRAM	Attempt to set fields in a null LOGINREC.
SYBEASTL	EXPROGRAM	Synchronous I/O attempted at AST level.

Error name	Error severity	Error text
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 in-stead!
SYBEBBCI	EXINFO	Batch successfully bulk copied to the server.
SYBEBBL	EXPROGRAM	Bad `<bindlen>` parameter passed to `dbsetnull`.
SYBEBCBC	EXPROGRAM	`bcp_columns` must be called before `bcp_colfmt` and `bcp_colfmt_ps`.
SYBEBCBNPR	EXPROGRAM	`bcp_bind`: if `<varaddr>` is NULL, `<prefixlen>` must be 0 and no terminator should be specified.
SYBEBCBNTYP	EXPROGRAM	`bcp_bind`: if `<varaddr>` is NULL and `<varlen>` greater than 0, the table column type must be SYB-TEXT or SYBIMAGE and the program variable type must be SYBTEXT, SYBCHAR, SYBIMAGE or SYBBI-NARY.
SYBEBCBPREF	EXPROGRAM	Illegal prefix length. Legal values are 0, 1, 2 or 4.
SYBEBCFO	EXUSER	`bcp` host files must contain at least one column.
SYBEBCHLEN	EXPROGRAM	`<host_collen>` 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 var-iables other than SYBCHAR, SYBBINARY, SYBTEXT, or SYBIMAGE.
SYBEBCITBLEN	EXPROGRAM	`bcp_init`: `<tblname>` parameter is too long.
SYBEBCITBNM	EXPROGRAM	`bcp_init`: `<tblname>` 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.

Error name	Error severity	Error text
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 `<table_colnum>` is 0, `<host_type>` 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.
SYBEBCSI	EXPROGRAM	Host-file columns may be skipped only when copy-ing into the Server.
SYBEBCSNDROW	EXPROGRAM	`bcp_sendrow` is not called unless all text data for the previous row has been sent using `bcp_moretext`.

Error name	Error severity	Error text
SYBEBCSNTYP	EXPROGRAM	column number %1!: If `<varaddr>` is NULL and `<varlen>` greater than 0, the table column type must be SYBTEXT or SYBIMAGE and the program variable type must be SYBTEXT, SYBCHAR, SYB-IMAGE 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	`<varlen>` 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 `<numbytes>` parameter passed to `dbstrcpy`.
SYBEBPKS	EXPROGRAM	In `DBSETLPACKET`, the packet size parameter must be between 0 and 999999.
SYBEBPREC	EXPROGRAM	Illegal precision specified.
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.

Error name	Error severity	Error text
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.
SYBEBTYPSRV	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.
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.

Error name	Error severity	Error text
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 data-stream.
SYBECDNS	EXCONSISTENCY	Datastream indicates that a compute column is de-rived 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 over-flow.
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.
SYBECRSBSKEY	EXPROGRAM	Keyset cannot be scrolled backward in mixed cur-sors 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`.

Error name	Error severity	Error text
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.
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.

Error name	Error severity	Error text
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.
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 `<datepart>` constant.

Error name	Error severity	Error text
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	SAP 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 `<computeid>` or compute column number.
SYBEIDCL	EXCONSISTENCY	Illegal datetime column length returned by Adaptive Server Enterprise. Legal datetime lengths are 4 bytes and 8 bytes.
SYBEIDECCL	EXCONSISTENCY	Invalid decimal column length returned by the server.
SYBEIFCL	EXCONSISTENCY	Illegal floating-point column length returned by Adaptive Server Enterprise. Legal floating-point lengths are 4 bytes 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 bytes and 8 bytes.

Error name	Error severity	Error text
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 DBPRO-CESSes lower than 1.
SYBEMVOR	EXPROGRAM	Month values must be between 1 and 12.
SYBENBUF	EXINFO	Called `dbsendpassthru` with a NULL `<buf>` parameter.
SYBENBVP	EXPROGRAM	Cannot pass `dbsetnull` a NULL `<bindval>` 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.

Error name	Error severity	Error text
SYBENMOB	EXPROGRAM	No such member of `order by` clause.
SYBENOEV	EXINFO	DBPOLL cannot 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.
SYBEOREN	EXINFO	International Release: Warning: an out-of-range error-number was encountered in `dblib.loc`. The maximum permissible error-number is defined as DBERRCOUNT in `sybdb.h`.
SYBEORPF	EXUSER	Attempt to set remote password would overflow the login record’s remote password field.
SYBEPOLL	EXINFO	There is already an active `dbpoll`.
SYBEPRTF	EXINFO	`dbtracestring` may only be called from a `<printfunc>`.
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.

Error name	Error severity	Error text
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 `<datalen>` 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	`<value>` parameter for `dbrpcparam` can be NULL, only if the `<datalen>` parameter is 0.
SYBERPTXTIM	EXPROGRAM	RPC parameters cannot be of type `text` or `image`.
SYBERPUL	EXPROGRAM	When passing a SYBINTN, SYBDATETIMN, SYBMONEYN, or SYBFLTN 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.

Error name	Error severity	Error text
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	SAP Adaptive Server Enterprise connection timed out.
SYBETMCF	EXPROGRAM	Attempt to install too many custom formats using dbfmtinstall.
SYBETMTD	EXPROGRAM	Attempt to send too much TEXT data using the dbmoretext call.
SYBETPAR	EXPROGRAM	No SYBTEXT or SYBIMAGE parameters were de-fined.
SYBETPTN	EXUSER	Syntax error: Only two periods are permitted in ta-ble 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 <numbytes> parameter passed to dbtracestring.
SYBETSIT	EXINFO	Attempt to call dbtsput with an invalid time-stamp.
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.

Error name	Error severity	Error text
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`.
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.

Error name	Error severity	Error text
SYBEWAID	EXCONSISTENCY	DB-Library internal error: ALTFMT following ALT-NAME 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 `<xlt_todisp>` parameter to `dbfree_xlate` was NULL. The space associated with the `xlt_tosrv` parameter has been freed.
SYBEXTN	EXPROGRAM	The `<xlt_tosrv>` and `<xlt_todisp>` parameters to `dbfree_xlate` were NULL.
SYBEXTSN	EXPROGRAM	Warning: The `<xlt_tosrv>` parameter to `dbfree_xlate` was NULL. The space associated with the `<xlt_todisp>` parameter has been freed.
SYBEZTXT	EXINFO	Attempt to send zero length `text` or `image` to dataserver using `dbwritetext`.
UNUSED	EXINFO	This error number is unused.

2.225 Options

The complete list of DB-Library options.

Syntax

```
#include <sybfront.h>

#include <sybdb.h>
```

Usage

- `dbsetopt` and `dbclropt` use the following constants, defined in `sybdb.h`, for setting and clearing options. All options are off by default. These options are available:

- DBARITHABORT – If this option is set, the server aborts a query when an arithmetic exception occurs during its execution.
- DBARITHIGNORE – If this option is set, the server substitutes null values for selected or updated values when an arithmetic exception occurs during query execution. The SAP Adaptive Server Enterprise (ASE) does not return a warning message. If neither DBARITHABORT nor DBARITHIGNORE is set, SAP ASE substitutes null values and print a warning message after the query has been executed.
- DBAUTH – This option sets system administration authorization levels. Possible levels are: "sa", "sso", "oper", and "dbcc_edit". For information on these levels, see the *SAP Adaptive Server Enterprise Reference Manual*.
- 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 is set to a default size (currently 1000 rows).
  When an application calls `dbclropt` to clear the DBBUFFER option, DB-Library frees the memory associated with the row buffer.
- DBCHAINXACTS – This option is used to select chained or unchained transaction behavior.
  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.
  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 DBPROCESSes, 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.

> **i Note**
>
> All DBPROCESSes opened using a particular LOGINREC also uses 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 `dbsqlexec` or `dbsqlsend` the first call to either `dbcmd` or `dbfcmd` 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 processes the query through the compile step but the query is not executed. This can be used 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) is 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 ASE'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 sends 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 reads 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. SAP ASE options can also be set through Transact-SQL commands.
- As mentioned in the preceding descriptions, certain options take parameters as:

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 `<int>` datatype is 2 bytes or 4 bytes long
DBTEXTLIMIT	"0" to "2,147,483,647"

`dbsetopt` requires that an option parameter is specified when setting any option on the preceding list. `dbclropt` 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`, `dbclropt`, and `dbisopt` are always passed as character strings and must be quoted, even if they are numeric values.

Related Information

dbgetrow [page 197]
dbnextrow [page 260]
dbclrbuf [page 107]
dbsetopt [page 385]
dbclropt [page 108]
dbisopt [page 207]
dbsetopt [page 385]
dbsetopt [page 385]

2.226 Types

Datatypes and symbolic constants for datatypes used by DB-Library.

Syntax

```
#include <sybfront.h>

#include <sybdb.h>
```

Usage

- 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.

  Symbolic Constants for Server Datatypes

Symbolic constant	Represents
SYBDATETIME	`datetime` type.
SYBDATETIME4	4-byte `datetime` type.

Symbolic constant	Represents
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. i Note Use DBCHAR as the type for program variables.
SYBSENSITIVITY	Security `sensitivity` type. i Note Use DBCHAR as the type for program variables.

See the *SAP 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 */
/* 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 datesecond;
/* 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;
```

i Note

The SYBBOUNDARY and SYBSENSITIVITY symbolic constants correspond to the program variable type DBCHAR.

Related Information

dbaltbind [page 68]

dbalttype [page 84]

dbbind [page 88]

dbcoltype [page 119]

dbconvert [page 124]

dbprtype [page 286]

dbrettype [page 331]

dbwillconvert [page 440]

Options [page 466]

3 Bulk Copy Routines

Review the information about the DB-Library bulk copy routines.

Related Information

Introduction to Bulk Copy [page 474]
List of Bulk Copy Routines [page 477]
bcp_batch [page 478]
bcp_bind [page 480]
bcp_colfmt [page 483]
bcp_colfmt_ps [page 487]
bcp_collen [page 491]
bcp_colptr [page 492]
bcp_columns [page 493]
bcp_control [page 494]
bcp_done [page 497]
bcp_exec [page 498]
bcp_getl [page 500]
bcp_init [page 501]
bcp_moretext [page 503]
bcp_options [page 506]
bcp_readfmt [page 507]
bcp_sendrow [page 508]
BCP_SETL [page 510]
bcp_setxlate [page 511]
bcp_writefmt [page 512]

3.1 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.

Related Information

Transferring Data into the Database [page 475]
Transferring Data out of the Database to a Flat File [page 476]

3.1.1 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.

Context

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 remains 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, SAP Adaptive Server Enterprise (ASE) 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 `bcp` 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:

1. Call `dblogin` to acquire a LOGINREC structure for later use with `dbopen`.
2. Call `BCP_SETL` to set up the LOGINREC for bulk copy operations into the database.
3. Call `dbopen` to establish a connection with the SAP ASE Server.
4. Call `bcp_init` to initialize the bulk copy operation and inform Adaptive Server Enterprise whether the copy is performed from program variables or from a host file. To copy data into the database, the `bcp_init` `<direction>` parameter must be passed as DB_IN.

At this point, an application copying data from program variables performs 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:

1. Call `bcp_bind` once for each program variable that is to be bound to a database column.
2. Transfer a batch of data in a loop:
    - Assign program variables the data values to transfer.
    - Call `bcp_sendrow` to send the row of data.

    - After a batch of rows has been sent, call `bcp_batch` to save the rows in SAP ASE Server.
3. 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 performs the following steps in addition to the introductory ones listed previously:

1. Call `bcp_control` to set the batch size and change control parameter default settings.
2. Call `bcp_columns` to set the total number of columns found in the host file.
3. 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`.
4. Call `bcp_exec` to start the copy in.

3.1.2 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.

Context

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 specific output file formats to be specified. Bulk copy is not significantly faster than SQL `select`.

Procedure

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.

3.2 List of Bulk Copy Routines

Review the description of bulk copy routines.

Routine	Description
bcp_batch	Save any preceding rows in SAP Adaptive Server Enterprise (ASE).
bcp_bind	Bind data from a program variable to a SAP ASE 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.
cp_done	End a bulk copy from program variables into SAP ASE.
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 SAP ASE.
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 SAP ASE.
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 SAP ASE.
bcp_writefmt	Write a datafile format definition to a host file.

Related Information

bcp_batch [page 478]

bcp_bind [page 480]
bcp_colfmt [page 483]
bcp_colfmt_ps [page 487]
bcp_collen [page 491]
bcp_colptr [page 492]
bcp_columns [page 493]
bcp_control [page 494]
bcp_done [page 497]
bcp_exec [page 498]
bcp_getl [page 500]
bcp_init [page 501]
bcp_moretext [page 503]
bcp_options [page 506]
bcp_readfmt [page 507]
bcp_sendrow [page 508]
BCP_SETL [page 510]
bcp_setxlate [page 511]
bcp_writefmt [page 512]

3.3 bcp_batch

Save any preceding rows in SAP Adaptive Server Enterprise.

Syntax

```
DBINT bcp_batch(dbproc)
DBPROCESS     *dbproc;
```

Parameters

dbproc

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

Returns

The number of rows saved since the last call to `bcp_batch`, or -1 in case of error.

Usage

- 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`.
- You may call `bcp_batch` once every `<n>` 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.
- By default, Adaptive Server Enterprise copies all the rows specified in one batch. SAP ASE 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—are copied into SAP Adaptive Server Enterprise.
- `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 are not saved. To find out whether this has happened, an application can refer to the messages generated by SAP ASE or DB-Library/C.

Related Information

bcp_bind [page 480]
bcp_done [page 497]
bcp_sendrow [page 508]

3.4 bcp_bind

Bind data from a program variable to an SAP Adaptive Server Enterprise table.

Syntax

```
RETCODE bcp_bind (dbproc, varaddr, prefixlen, varlen,
                terminator, termlen, type,
                table_column)

DBPROCESS     *dbproc;
BYTE                  *varaddr;
int                         prefixlen;
DBINT                  varlen;
BYTE                  *terminator;
int                         termlen;
int                         type;
int                         table_column;
```

Parameters

dbproc

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

varaddr

The address of the program variable from which the data is copied. If type is SYBTEXT or SYBIMAGE, `<varaddr>` can be NULL. A NULL `<varaddr>` indicates that `text` and `image` values are sent to SAP ASE in chunks by `bcp_moretext`, rather than all at once by `bcp_sendrow`.

prefixlen

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 `<prefixlen>` to 0.

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 uses either a length prefix or a terminator sequence to determine the length. (If both are supplied, the system uses 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 returns 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 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 SAP ASE datatype. The data in the program variable is converted to the type of the database column. If this parameter is 0, no conversion is performed. For a list of supported conversions and a list of SAP ASE datatypes, see dbconvert [page 124]

table_column

The column in the database table to which the data is copied. Column numbers start at 1.

Returns

SUCCEED or FAIL.

Examples

Example 1

- 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);
dbmsghandle(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_collen.
- 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 results 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.

Related Information

dbconvert [page 124]
bcp_batch [page 478]
bcp_colfmt [page 483]
bcp_collen [page 491]
bcp_colptr [page 492]
bcp_columns [page 493]
bcp_control [page 494]
bcp_done [page 497]
bcp_exec [page 498]
bcp_init [page 501]
bcp_moretext [page 503]
bcp_sendrow [page 508]

3.5 bcp_colfmt

Specify the format of a host file for bulk copy purposes.

Syntax

```
RETCODE bcp_colfmt (dbproc, host_colnum, host_type,
                    host_prefixlen, host_collen,
                    host_term, host_termlen,
                    table_colnum)

DBPROCESS      *dbproc;
```

```
int                         host_colnum;
int                         host_type;
int                         host_prefixlen;
DBINT                  host_collen;
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/ SAP Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and SAP ASE.

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 SAP ASE datatype. If it is different from the datatype of the corresponding column in the database table (`<table_colnum>`), the conversion is performed. For a list of supported conversions and a list of SAP ASE datatypes, see dbconvert [page 124]

If you want to specify the same datatype as in the corresponding column of the database table (`<table_colnum>`), set this parameter to 0.

> **i 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, set this parameter to 0. To let `bcp` decide whether to use a length prefix, set this parameter to -1. In such a case, `bcp` 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` looks 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 that you have a 1-byte length prefix for a 4-byte integer column. Ordinarily, the length prefix contains 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 uses either a length prefix or a terminator sequence to determine the length of the data. (If both are supplied, the system uses 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 returns an error message. If `<host_collen>` is 0, the system assumes that 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` looks 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 is not copied. The first column is column 1.

Returns

SUCCEED or FAIL.

Usage

- `bcp_colfmt` 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` 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, first call `bcp_columns(<dbproc>`, 5), and then call `bcp_colfmt` 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_collen>`, and `<host_termlen>` 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`.
- You must call `bcp_colfmt` 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.

Related Information

dbconvert [page 124]
bcp_batch [page 478]
bcp_bind [page 480]
bcp_colfmt_ps [page 487]
bcp_collen [page 491]
bcp_colptr [page 492]
bcp_columns [page 493]
bcp_control [page 494]
bcp_done [page 497]
bcp_exec [page 498]
bcp_init [page 501]
bcp_sendrow [page 508]

3.6 bcp_colfmt_ps

Specify the format of a host file for bulk copy purposes, with precision and scale support for `numeric` and `decimal` columns.

Syntax

```
RETCODE bcp_colfmt_ps (dbproc, host_colnum, host_type,
                     host_prefixlen, host_collen,
                     host_term, host_termlen,
                     table_colnum, typeinfo)

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;
```

i 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/ SAP Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and SAP ASE.

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 SAP ASE datatype. If it is different from the datatype of the corresponding column in the database table (`<table_colnum>`), the conversion is performed. For a list of supported conversions and a list of SAP ASE datatypes, see dbconvert [page 124]

If you want to specify the same datatype as in the corresponding column of the database table (`<table_colnum>`), set this parameter 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, set this parameter to 0. To let `bcp` decide whether to use a length prefix, set this parameter to -1. In such a case, `bcp` uses 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` looks 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 that you have a 1-byte length prefix for a 4-byte integer column. Ordinarily, the length prefix contains 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 uses either a length prefix or a terminator sequence to determine the length of the data. (If both are supplied, the system uses 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 returns an error message. If `<host_collen>` is 0, the system assumes that 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` looks 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 is not 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>`.

Returns

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, 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_collen>, and <host_termlen> 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.

Related Information

dbconvert [page 124]
bcp_batch [page 478]
bcp_bind [page 480]
bcp_colfmt [page 483]
bcp_collen [page 491]
bcp_colptr [page 492]
bcp_columns [page 493]
bcp_control [page 494]
bcp_done [page 497]
bcp_exec [page 498]
bcp_init [page 501]
bcp_sendrow [page 508]

3.7 bcp_collen

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/ SAP Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and SAP ASE.

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 is determined by the length prefix or terminator. If both a length prefix and a terminator exist, `bcp` uses 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 is copied. Column numbers start at 1.

Returns

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` uses 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.

Related Information

bcp_bind [page 480]
bcp_colptr [page 492]
bcp_sendrow [page 508]

3.8 bcp_colptr

Set the program variable data address for the current bulk copy into the database.

Syntax

```
RETCODE bcp_colptr(dbproc, colptr, table_column)

DBPROCESS     *dbproc;
BYTE                  *colptr;
int                         table_column;
```

Parameters

dbproc

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

colptr

The address of the program variable.

table_column

The column in the SAP ASE table to which the data is copied. Column numbers start at 1.

Returns

SUCCEED or FAIL.

Usage

- The `bcp_colptr` function allows you to change the program variable data address for a particular column while running a copy `in` through calls to `bcp_bind.`
- Initially, the program variable data address is determined when `bcp_bind` is called. If the program variable data address changes between calls to `bcp_sendrow,` you may call `bcp_colptr` to reset the address of the data. The next call to `bcp_sendrow` uses the data at the address you just set.
- There must be a separate `bcp_colptr` call for every column in the table whose data address you want to modify.

Related Information

bcp_bind [page 480]
bcp_collen [page 491]
bcp_sendrow [page 508]

3.9 bcp_columns

Set the total number of columns found in the host file.

Syntax

```
RETCODE bcp_columns(dbproc, host_colcount)

DBPROCESS     *dbproc;
int                         host_colcount;
```

Parameters

dbproc

A pointer to the DBPROCESS structure that provides the connection for a particular front-end/ SAP Adaptive Server Enterprise process. It contains all the information that

DB-Library uses to manage communications and data between the front end and SAP ASE.

host_colcount

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 `<host_colcount>` to the total number of host file columns.

Returns

SUCCEED or FAIL.

Usage

- 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.
- 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.
- After calling `bcp_columns`, you must call `bcp_colfmt` `<host_colcount>` times, because you are defining a completely custom file format.

Related Information

bcp_init [page 501]
bcp_colfmt [page 483]
bcp_init [page 501]

3.10 bcp_control

Change various control parameter default settings.

Syntax

```
RETCODE bcp_control(dbproc, field, value)

DBPROCESS     *dbproc;
```

```
int                           field;
DBINT                     value;
```

Parameters

dbproc

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

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 is done in one batch. This field is only meaningful when copying from a host file into SAP ASE.

value

The value to change the corresponding control parameter to.

Returns

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);
}
```

Related Information

bcp_batch [page 478]

bcp_bind [page 480]

bcp_colfmt [page 483]

bcp_collen [page 491]

bcp_colptr [page 492]

bcp_columns [page 493]
bcp_done [page 497]
bcp_exec [page 498]
bcp_init [page 501]

3.11 bcp_done

End a bulk copy from program variables into SAP 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/SAP ASE process. It contains all the information that DB-Library uses to manage communications and data between the front end and SAP ASE.

Returns

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 results in unpredictable errors.

Related Information

bcp_batch [page 478]
bcp_bind [page 480]
bcp_moretext [page 503]
bcp_sendrow [page 508]

3.12 bcp_exec

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/ SAP Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and SAP ASE.

rows_copied

A pointer to a DBINT. `bcp_exec` fills this DBINT with the number of rows successfully copied. If set to NULL, this parameter is not filled in by `bcp_exec`.

Returns

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 `<rows_copied>` parameter for the number of rows successfully copied.

Usage

- This routine copies data from a host file to a database table or vice versa, depending on the value of the `<direction>` parameter in `bcp_init`.
- Before calling this function you must call `bcp_init` with a valid host file name. Failure to do so results in an error.
- The following program fragment illustrates `bcp_exec`:

```
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');
```

Related Information

bcp_batch [page 478]

bcp_bind [page 480]

bcp_colfmt [page 483]

bcp_collen [page 491]

bcp_colptr [page 492]

bcp_columns [page 493]

bcp_control [page 494]

bcp_done [page 497]

bcp_init [page 501]

bcp_sendrow [page 508]

3.13 bcp_getl

Determine if the LOGINREC has been set for bulk copy operations.

Syntax

```
DBBOOL bcp_getl(loginrec)

LOGINREC        *loginrec;
```

Parameters

loginrec

A pointer to a LOGINREC structure that is passed as an argument to `dbopen`. You can get a LOGINREC structure by calling `dblogin`.

Returns

“TRUE” or “FALSE.”

Usage

- `bcp_getl` returns “TRUE” if *`<loginrec>` is enabled for bulk copy operations, and “FALSE” if it is not.
- 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.
- 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 `<enable>` 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.
- If LOGINREC is NULL, `bcp_getl` returns “FALSE.”

Related Information

bcp_init [page 501]

BCP_SETL [page 510]
dblogin [page 213]
dbopen [page 274]

3.14 bcp_init

Initialize bulk copy.

Syntax

```
RETCODE bcp_init(dbproc, tblname, hfile, errfile,
               direction)

DBPROCESS     *dbproc;
char                    *tblname;
char                    *hfile;
char                    *errfile;
int                         direction;
```

Parameters

dbproc

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

tblname

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.

hfile

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), `<hfile>` should be NULL.

errfile

The name of the error file to be used. This error file is 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 SAP ASE table.

If `<errfile>` is NULL, no error file is used.

If `<hfile>` is NULL, `<errfile>` is ignored. This is because an error file is not necessary when bulk-copying from program variables.

direction

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.

It is illegal to request a bulk copy from the database table (DB_OUT) without supplying a host file name.

Returns

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 results in an error.
- If a host file is being used (see the description of `<hfile>` in the preceding “Parameters” section), 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 is also fixed-length. If a given database column’s data is variable-length or may contain null values, the host file’s data column is 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 does the following:

    ```
    sp_dboption ’mydb’, ’select into/bulkcopy’, ’true’
    ```

  See the *SAP 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.

Related Information

bcp_batch [page 478]
bcp_bind [page 480]
bcp_colfmt [page 483]
bcp_collen [page 491]
bcp_colptr [page 492]
bcp_columns [page 493]
bcp_control [page 494]
bcp_done [page 497]
bcp_exec [page 498]
bcp_sendrow [page 508]

3.15 bcp_moretext

Send part of a `text` or `image` value to SAP 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/ SAP ASE process. It contains all the information that DB-Library uses to manage communications and data between the front end and SAP ASE.

size

The size of this particular part of the `text` or `image` value being sent to SAP ASE. It is an error to send more `text` or `image` bytes to SAP ASE than were specified in the call to `bcp_bind` or `bcp_collen`.

text

A pointer to the `text` or `image` portion to be written.

Returns

SUCCEED or FAIL.

Usage

- This routine is used 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 long data buffers.
- If `bcp_bind` is called with a `<type>` parameter of SYBTEXT or SYBIMAGE and a non-NULL `<varaddr>` parameter, `bcp_sendrow` sends 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` returns control to the application immediately after all non-`text` or `image` columns are sent to SAP ASE. The application can then call `bcp_moretext` repeatedly to send the `text` and `image` columns to SAP ASE, 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` first sends 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 calls `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(...);
```

Related Information

bcp_bind [page 480]
bcp_sendrow [page 508]
dbmoretext [page 254]
dbwritetext [page 443]

3.16 bcp_options

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/ SAP Adaptive Server Enterprise process. It contains all the information that DB-Library uses to manage communications and data between the front end and SAP ASE.

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>`:

Values for Value (bcp_options)

If `<option >`Is	**`<*value>` should be**
BCPLABELED	A DBBOOL value. Set `<*value>` to “true” to allow a bulk copy with sensitivity labels. Set `<*value>` 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.

Returns

SUCCEED or FAIL.

Usage

- `bcp_options` sets bulk copy options.
- Currently the only bulk copy option available is BCPLABELED.

Related Information

bcp_init [page 501]
bcp_control [page 494]

3.17 bcp_readfmt

Read a datafile format definition from a host file.

Syntax

```
RETCODE bcp_readfmt(dbproc, filename)

DBPROCESS    *dbproc;
char                  *filename;
```

Parameters

dbproc

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

filename

The full directory specification of the file containing the format definitions.

Returns

SUCCEED or FAIL.

Usage

- 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.
- 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 *Open Client and Open Server Programmers Supplement*.
- Application programs can call bcp_writefmt to create files with datafile format definitions.
- The following code fragment illustrates the use of bcp_readfmt:

```
bcp_init(dbproc, "mytable", "bcpdata", "bcperrs", DB_IN);
bcp_readfmt(dbproc, "my_fmtfile");
bcp_exec(dbproc, &rows_copied);
```

Related Information

bcp_colfmt [page 483]
bcp_columns [page 493]
bcp_writefmt [page 512]

3.18 bcp_sendrow

Send a row of data from program variables to SAP Adaptive Server Enterprise.

Syntax

```
RETCODE bcp_sendrow(dbproc)

DBPROCESS      *dbproc;
```

Parameters

dbproc

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

Returns

SUCCEED or FAIL.

Usage

- `bcp_sendrow` builds a row from program variables and sends it to Adaptive Server Enterprise.
- Before calling `bcp_sendrow`, you must make calls to `bcp_bind` to specify the program variables to be used.
- If `bcp_bind` is called with a `<type>` parameter of SYBTEXT or SYBIMAGE and a non-null `<varaddr>` parameter, `bcp_sendrow` sends 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` returns control to the application immediately after all non-`text` or `image` columns are sent to SAP ASE. The application can then call `bcp_moretext` repeatedly to send the `text` and `image` columns to SAP ASE, a chunk at a time. For an example, see bcp_moretext [page 503].
- After the last call to `bcp_sendrow`, you must call `bcp_done` to ensure proper internal cleanup.
- When `bcp_sendrow` is used to bulk copy rows from program variables into Adaptive Server Enterprise tables, rows are permanently saved in SAP ASE only when the user calls `bcp_batch` or `bcp_done`. The user may choose to call `bcp_batch` once every `<n>` 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 `bcp_batch` at all. If `bcp_batch` is never called, the rows are permanently saved in SAP ASE when `bcp_done` is called.

Related Information

bcp_moretext [page 503]
bcp_batch [page 478]
bcp_bind [page 480]
bcp_colfmt [page 483]
bcp_collen [page 491]
bcp_colptr [page 492]
bcp_columns [page 493]
bcp_control [page 494]
bcp_done [page 497]
bcp_exec [page 498]
bcp_init [page 501]
bcp_moretext [page 503]

3.19 BCP_SETL

Set the LOGINREC for bulk copy operations into the database.

Syntax

```
RETCODE BCP_SETL(loginrec, enable)

LOGINREC      *loginrec;
DBBOOL          enable;
```

Parameters

loginrec

This is a pointer to a LOGINREC structure, which is passed as an argument to `dbopen`. You can get a LOGINREC structure by calling `dblogin`.

enable

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 not enabled for bulk copy operations.

Returns

SUCCEED or FAIL.

Usage

- 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.
- 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 `<enable>` 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.
- `BCP_SETL` applies to “copy in” operations only.

Related Information

bcp_init [page 501]
bcp_getl [page 500]
dblogin [page 213]
dbopen [page 274]
DBSETLAPP [page 365]
DBSETLHOST [page 369]
DBSETLPWD [page 378]
DBSETLUSER [page 380]

3.20 bcp_setxlate

Specify the character set translations to use when retrieving data from or inserting data into an SAP Adaptive Server Enterprise.

Syntax

```
RETCODE bcp_setxlate(dbproc, xlt_tosrv, xlt_todisp)

DBPROCESS    *dbproc;
DBXLATE          *xlt_tosrv;
DBXLATE          *xlt_todisp;
```

Parameters

dbproc

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

xlt_tosrv

A pointer to a translation structure. The translation structure is allocated using `dbload_xlate`. `<xlt_tosrv>` indicates the character set translation to use when moving data from the application program to the SAP ASE (the copy in, or DB_IN, direction).

xlt_todisp

A pointer to a translation structure. The translation structure is allocated using `dbload_xlate`. `<xlt_todisp>` indicates the character set translation to use when moving data from SAP ASE to the application program (the copy out, or DB_OUT, direction).

Returns

SUCCEED or FAIL.

Usage

- `bcp_setxlate` specifies the character set translations to use when transferring character data between the SAP ASE and a front-end application program using `bcp`.
- 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.
- The following code fragment illustrates the use of `bcp_setxlate`:

```
bcp_init(dbproc, "mytable", "bcpdata", "bcperrs", 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);
```

Related Information

dbfree_xlate [page 181]
dbload_xlate [page 210]
dbxlate [page 448]

3.21 bcp_writefmt

Write a datafile format definition to a host file.

Syntax

```
RETCODE bcp_writefmt(dbproc, filename)
```

```
DBPROCESS      *dbproc;
char                     *filename;
```

Parameters

dbproc

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

filename

The full directory specification of the file that contains the format definitions.

Returns

SUCCEED or FAIL.

Usage

- `bcp_writefmt` writes a datafile format definition to a host file. The format reflects previous calls to `bcp_columns` and `bcp_colfmt`.
- `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 *Open Client and Open Server Programmers Supplement*.
- Format definition files are read using `bcp_readfmt`.
- The following code fragment illustrates the use of `bcp_writefmt`:

```
bcp_init(dbproc, "mytable", "bcpdata", "bcperrs", DB_OUT);
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_writefmt(dbproc, "my_fmtfile");
bcp_exec(dbproc, &rows_copied);
```

Related Information

bcp_colfmt [page 483]

bcp_columns [page 493]

bcp_readfmt [page 507]

4 Two-Phase Commit Service

SAP Adaptive Server Enterprise provides a two-phase commit service that allows a client application to coordinate transactions that are distributed on two or more SAP ASEs.

The two-phase commit process and the DB-Library routines that are involved.

Related Information

Programming Distributed Transactions [page 515]
Commit Service and the Application Program [page 516]
The Probe Process [page 517]
Two-Phase Commit Routines [page 517]
Specifying the Commit Server [page 519]
Two-Phase Commit Sample Program [page 519]
Program Notes [page 524]
abort_xact [page 527]
build_xact_string [page 528]
close_commit [page 530]
commit_xact [page 531]
open_commit [page 532]
remove_xact [page 533]
scan_xact [page 534]
start_xact [page 535]
stat_xact [page 537]

4.1 Programming Distributed Transactions

Review the information about how distributed transactions take place.

Context

The two-phase commit service allows an application to coordinate updates among two or more SAP Adaptive Server Enterprises. This initial implementation of distributed transactions treats separate transactions (which may be on separate SAP ASE) as if they were a single transaction. The commit service uses one SAP ASE, 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 SAP ASEs 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 takes 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.

> **i Note**
>
> If certain types of errors occur during a two-phase transaction, SAP ASE 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 SAP ASE is able to mark processes as infected, boot SAP ASE with the flag `-T3620` passed on the command line.

4.2 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 SAP ASEs 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 SAP ASE or the application program fails before the `prepare transaction` statement, the SAP ASE will `rollback` the transaction.

If any SAP ASE or the application program fails after the `prepare` but before the `commit`, the SAP ASE communicates with the server functioning as the commit service and ask it whether to `rollback` or `commit`.

If the SAP ASE cannot communicate with the server functioning as the commit service, it marks the user task process as infected in SAP ASE. 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 SAP ASE communicates 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 SAP ASE communicates 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.

4.3 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.

4.4 Two-Phase Commit Routines

Review the routines that make up the two-phase commit service.

Routine	Description
abort_xact	Tells the commit service to abort the transaction.
build_xact_string	Builds a name string for use by each participating SAP 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 `<commid>`.
close_commit	Closes the connection with the commit service.
commit_xact	Tells the commit service to commit the transaction.

Routine	Description
open_commit	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 DBPROCESS structure used in subsequent commit service calls.
remove_xact	Decrements the count of servers still participating in the transaction.
start_xact	Records the start of a distributed transaction and stores initial information about the transaction (DBPROCESS id, application name, transaction name, and number of sites participating) in a lookup table on the commit server. It returns the `<commid>` identifying number for the transaction.

Two additional routines are used for ongoing status reports:

Routine	Description
scan_xact	Returns the status of a single transaction or all distributed transactions.
stat_xact	Returns the completion status of a distributed transaction.

During 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.

Related Information

abort_xact [page 527]
build_xact_string [page 528]
close_commit [page 530]
commit_xact [page 531]

open_commit [page 532]
remove_xact [page 533]
start_xact [page 535]
scan_xact [page 534]
stat_xact [page 537]

4.5 Specifying the Commit Server

Review the instructions to specify the commit server/.

Context

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 (:).

4.6 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);
dbsqlexec(dbproc_server2);
/*
** Start the distributed transaction on the
** commit service.
*/
commid = start_xact(dbproc_commit, "demo", "test", 2);
```

i 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);
dbsqlexec(dbproc_server1);
dbcmd(dbproc_server2, cmdbuf);
dbsqlexec(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 = dbsqlexec(dbproc_server1);
dbcmd(dbproc_server2, cmdbuf);
ret_server2 = dbsqlexec(dbproc_server2);
```

```
 /*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);
dbsqlexec(dbproc_server1);
dbcmd(dbproc_server2, cmdbuf);
dbsqlexec(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 = dbsqlexec(dbproc_server1);
dbcmd(dbproc_server2, cmdbuf);
ret_server2 = dbsqlexec(dbproc_server2);
```

i Note

See Program Note 1 [page 525].

```
     if (ret_server1 == FAIL || ret_server2 == FAIL)
{
/* Some part of the transaction failed. */
printf(" Transaction aborted -- dbsqlexec 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);
```

i Note

See Program Note 2 [page 525].

```
     ret_server2 = dbsqlexec(dbproc_server2);
```

i Note

See Program Note 3 [page 525].

```
if (ret_server1 == FAIL || ret_server2 == FAIL)
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);
}
```

i Note

See Program Note 4 [page 525].

```
    /* 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);
}
```

i Note

See Program Note 5 [page 526].

```
     /* 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);
```

i Note

See Program Note 6 [page 526].

```
      dbcmd(dbproc_server2, cmdbuf);
if (dbsqlexec(dbproc_server2) != FAIL)
remove_xact(dbproc_commit, commid, 1);
```

i Note

See Program Note 7 [page 527].

```
   /* Close the connection to the commit server. */
close_commit(dbproc_commit);
```

i Note

See Program Note 8 [page 527].

```
  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 (dbsqlexec(dbproc_server1) != FAIL)
remove_xact(dbproc_commit, commid, 1);
dbcmd(dbproc_server2, cmdbuf);
if (dbsqlexec(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);
}
```

Related Information

Program Note 1 [page 525]

Program Note 2 [page 525]

Program Note 3 [page 525]

Program Note 4 [page 525]

Program Note 5 [page 526]

Program Note 6 [page 526]

Program Note 7 [page 527]

Program Note 8 [page 527]

4.7 Program Notes

Review the program notes in the sample code.

Related Information

Program Note 1 [page 525]

Program Note 2 [page 525]

Program Note 3 [page 525]

Program Note 4 [page 525]

Program Note 5 [page 526]
Program Note 6 [page 526]
Program Note 7 [page 527]
Program Note 8 [page 527]

4.7.1 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`.

4.7.2 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.

4.7.3 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`.

4.7.4 Program Note 4

Review the information of 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` succeeds, but the application's `commit transaction` command to the participating server will not. However, the server is aware that its connection with the application has died. It communicates 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` succeeds, but the application's `commit transaction` to the participating server does not. When the participating server comes back up, it uses `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 is not able to communicate with the participating server. The participating server uses `probe` to communicate with the commit server. It learns that the transaction has not been committed in the commit service, and rolls 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` succeeds, but the application cannot communicate this to the participating server. When its connection to the application dies, the participating server attempts 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.

4.7.5 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 is told to commit the transaction locally. After this point, the application does not need to concern itself with aborting the transaction.

4.7.6 Program Note 6

If the above `dbsqlexec` to Server1 fails because the application's link to the server has gone down, Server1 uses `probe` to communicate with the commit server. `probe` finds that the transaction is committed in the commit server and tells Server1 to commit locally.

If `probe` cannot communicate with the commit server, Server1 infects the user task process in SAP Adaptive Server Enterprise. If the System Administrator kills the infected process before communication with the commit server is reestablished, the transaction is 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 `dbsqlexec` to Server1 fails because Server1 has gone down, the local transaction remains in a suspended state until Server1 is restored. As part of the recovery process, Server1 uses `probe` to communicate with the commit server. `probe` finds that the transaction is committed in the commit server and tells Server1 to commit locally.

If `probe` cannot communicate with the commit server, Server1 marks the database as suspect. After communication with the commit sever is reestablished, the suspect database should be re-recovered.

4.7.7 Program Note 7

If the above dbsqlexec 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 dbsqlexec 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.

4.7.8 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.

4.8 abort_xact

Mark a distributed transaction as being aborted.

Syntax

```
RETCODE abort_xact(connect, commid)

DBPROCESS      *connect;
DBINT                  commid;
```

Parameters

connect

A pointer to the DBPROCESS used to communicate with the commit service.

commid

The <commid> used to identify the transaction to the commit service.

Returns

SUCCEED or FAIL.

Usage

This routine informs the commit service that the status of a distributed transaction should be changed from "begin" to "abort."

Related Information

commit_xact [page 531]
remove_xact [page 533]
scan_xact [page 534]
start_xact [page 535]
stat_xact [page 537]

4.9 build_xact_string

Build a name for a distributed transaction.

Syntax

```
void build_xact_string(xact_name, service_name,
                     commid, result)

char          *xact_name;
char          *service_name;
DBINT       commid;
char         * result;
```

Parameters

xact_name

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.

service_name

The name that is used by SAP 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.

Returns

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 SAP ASE 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.

Related Information

commit_xact [page 531]
start_xact [page 535]

4.10 close_commit

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`.

Returns

None.

Usage

This routine calls `dbclose` to end a connection with the commit service. A call to `close_commit` should be made when the application is through with the commit service, to free resources.

Related Information

dbclose [page 106]

4.11 commit_xact

Mark a distributed transaction as being committed.

Syntax

```
RETCODE commit_xact(connect, commid)

DBPROCESS     *connect;
DBINT                    commid;
```

Parameters

connect
: A pointer to the DBPROCESS used to communicate with the commit service.

commid
: The `<commid>` used to identify the transaction to the commit service.

Returns

SUCCEED or FAIL.

If `commit_xact` fails, 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."

Related Information

abort_xact [page 527]
remove_xact [page 533]
scan_xact [page 534]
start_xact [page 535]

stat_xact [page 537]

4.12 open_commit

Establish a connection with the commit service.

Syntax

```
DBPROCESS *open_commit(login, servername)

LOGINREC      *login;
char                 *servername;
```

Parameters

login

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 `dblogin`. The caller may wish to initialize fields in the LOGINREC. For more details, see dblogin [page 213].

servername

The name of the commit service; for example, DSCOMMIT_SALESNET. If `<servername>` is NULL, the name DSCOMMIT is used. The name cannot contain a period (.) or a colon (:).

Returns

A pointer to a DBPROCESS structure to be used in subsequent commit service calls, or NULL if the open failed.

Usage

- This routine calls `dbopen` to establish a connection with the commit service. A call to `open_commit` must precede any calls to other commit service routines, such as `start_xact`, `commit_xact`, `abort_xact`, `remove_xact`, and `scan_xact`. A session with the commit service is closed by calling `close_commit`.

- 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.

Related Information

dblogin [page 213]
dblogin [page 213]
dbopen [page 274]

4.13 remove_xact

Decrement the count of sites still active in the distributed transaction.

Syntax

```
RETCODE remove_xact(connect, commid, n)

DBPROCESS      *connect;
DBINT             commid;
int                    n;
```

Parameters

connect
: A pointer to the DBPROCESS used to communicate with the commit service.

commid
: The `<commid>` used to identify the transaction to the commit service.

n
: The number of sites to remove from the transaction.

Returns

SUCCEED or FAIL.

Usage

- 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.
- The transaction record is deleted entirely if the count drops to 0.

Related Information

abort_xact [page 527]
commit_xact [page 531]
scan_xact [page 534]
start_xact [page 535]
stat_xact [page 537]

4.14 scan_xact

Print commit service record for distributed transactions.

Syntax

```
RETCODE scan_xact(connect, commid)

DBPROCESS      *connect;
DBINT                    commid;
```

Parameters

connect

A pointer to the DBPROCESS used to communicate with the commit service.

commid

The `<commid>` used to identify the transaction to the commit service. If `<commid>` is -1, all commit service records are displayed.

Returns

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.

Related Information

abort_xact [page 527]
commit_xact [page 531]
remove_xact [page 533]
start_xact [page 535]
stat_xact [page 537]

4.15 start_xact

Start a distributed transaction using the commit service.

Syntax

```
DBINT start_xact(connect, application_name, xact_name,
               site_count)

DBPROCESS     *connect;
char                    *application_name;
char                    *xact_name;
int                         site_count;
```

Parameters

connect
    A pointer to the DBPROCESS used to communicate with the commit service.

application_name

The name of the application. The application developer can choose any name for the application. It appears in the table maintained by the commit service but is not used by the commit service or the SAP Adaptive Server Enterprise recovery system.

xact_name

The name of the transaction. This name appears in the table maintained by the commit service and must be supplied as part of the transaction name string built by `build_xact_string`. The name cannot contain a period (.) or a colon (:).

site_count

The number of sites participating in the transaction.

Returns

An integer called the `<commid>`. This number is used to identify the transaction in subsequent calls to the commit service. In case of error, this routine returns 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 `<commid>`, which is a number that caller then uses to identify the transaction to the commit service.

Related Information

abort_xact [page 527]
build_xact_string [page 528]
commit_xact [page 531]
remove_xact [page 533]
scan_xact [page 534]
stat_xact [page 537]

4.16 stat_xact

Return the current status of a distributed transaction.

Syntax

```
int stat_xact(connect, commid)

DBPROCESS     *connect;
DBINT                   commid;
```

Parameters

connect

A pointer to the DBPROCESS used to communicate with the commit service.

commid

The `<commid>` is used to identify the transaction to the commit service. If `<commid>` is -1, all commit service records are displayed.

Returns

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.

Related Information

abort_xact [page 527]
commit_xact [page 531]
remove_xact [page 533]
scan_xact [page 534]
start_xact [page 535]

5 Cursors

Review the information about the DB-Library cursor.

Related Information

Cursor Overview [page 538]
Sensitivity to Change [page 540]
DB-Library Cursor Functions [page 543]
Holding Locks [page 543]
Stored Procedures Used by DB-Library Cursors [page 544]

5.1 Cursor Overview

Review the information about the cursor.

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 SAP 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.

Related Information

DB-Library Cursor Capability [page 539]
Differences Between DB-Library Cursors and Browse Mode [page 539]
Differences Between DB-Library and Client-Library Cursors [page 539]

5.1.1 DB-Library Cursor Capability

Review the capabilities of DB-Library cursor routines.

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

5.1.2 Differences Between DB-Library Cursors and Browse Mode

Review the difference between DB-Library cursors and browse mode.

Cursors let the user scroll through and update a result set with fewer restrictions than 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:

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

5.1.3 Differences Between DB-Library and Client-Library Cursors

Review the difference between DB-Library cursors and Client-Library cursors.

A DB-Library cursor does not correspond to an actual SAP Adaptive Server Enterprise cursor. Instead, at the time the cursor is declared with `dbcursoropen`, DB-Library fetches keysets from SAP ASE “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:

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 `<scrollopt>` during `dbcursoropen`).	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 `dbcursoropen`.	More memory is not required, regardless of how large the row sizes are.
You cannot access an SAP Open Server application unless the application installs the required DB-Library stored procedures.	You can access a version 10.0 (or later) SAP Open Server application that is coded to support cursors.
Slower performance.	Faster performance.

5.2 Sensitivity to Change

Three broad categories identify cursors according to their sensitivity to change.

They are:

- 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.

Related Information

Static Cursor [page 541]
Keyset-Driven Cursor [page 541]
Dynamic Cursor [page 541]
Concurrency Control [page 542]

5.2.1 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.

5.2.2 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.

5.2.3 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.

5.2.4 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:

Concurrency control options

Option	Result
CUR_READONLY	Updates are not permitted.
CUR_LOCKCC	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. 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.
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:

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.

5.3 DB-Library Cursor Functions

Review the list that 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.
dbcursorclose	Closes the cursor.

For details about an individual routine, see its reference page.

Related Information

dbcursoropen [page 148]
dbcursorinfo [page 147]
dbcursorcolinfo [page 143]
dbcursorbind [page 139]
dbcursorfetch [page 144]
dbcursor [page 137]
dbcursorclose [page 142]

5.4 Holding Locks

Review the instructions to hold locks.

Context

To retain the flexibility of the SAP 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 `dbcursoropen`, the application must issue `commit transaction` and `begin transaction` before each `dbcursorfetch` that scrolls the local buffer (except for the very first `dbcursorfetch`, 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 `dbcursor`. 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 `dbcursoropen`, and issue `begin transaction` before the first `dbcursorfetch`. 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.

5.5 Stored Procedures Used by DB-Library Cursors

DB-Library's cursor routines call the SAP Adaptive Server Enterprise's catalog stored procedures to find out table formats and identify key columns.

See the *SAP Adaptive Server Enterprise Reference Manual*.

6 DB-Library Error Messages

Review the error messages given by DB-Library.

Related Information

20001 [page 552]
20002 [page 553]
20003 [page 554]
20004 [page 555]
20005 [page 556]
20006 [page 557]
20008 [page 558]
20009 [page 559]
20010 [page 560]
20011 [page 561]
20012 [page 562]
20013 [page 563]
20014 [page 564]
20015 [page 565]
20016 [page 566]
20017 [page 567]
20018 [page 568]
20019 [page 569]
20020 [page 570]
20021 [page 571]
20022 [page 572]
20023 [page 573]
20024 [page 574]
20025 [page 575]
20026 [page 576]
20027 [page 577]
20028 [page 578]
20029 [page 579]
20030 [page 580]
20031 [page 581]
20033 [page 582]
20034 [page 583]
20035 [page 584]

20036 [page 585]
20037 [page 586]
20038 [page 587]
20039 [page 588]
20040 [page 589]
20041 [page 590]
20042 [page 591]
20043 [page 592]
20044 [page 593]
20045 [page 594]
20046 [page 595]
20047 [page 596]
20048 [page 597]
20049 [page 598]
20050 [page 599]
20051 [page 600]
20052 [page 601]
20053 [page 602]
20054 [page 603]
20055 [page 604]
20056 [page 605]
20060 [page 606]
20061 [page 607]
20062 [page 608]
20063 [page 609]
20064 [page 610]
20065 [page 611]
20066 [page 612]
20067 [page 613]
20068 [page 614]
20069 [page 615]
20070 [page 616]
20071 [page 617]
20072 [page 618]
20073 [page 619]
20074 [page 620]
20075 [page 621]
20076 [page 622]
20077 [page 623]
20078 [page 624]
20079 [page 625]
20080 [page 626]
20081 [page 627]

20082 [page 628]
20083 [page 629]
20084 [page 630]
20085 [page 631]
20086 [page 632]
20087 [page 633]
20088 [page 634]
20091 [page 635]
20092 [page 636]
20093 [page 637]
20094 [page 638]
20095 [page 639]
20096 [page 640]
20097 [page 641]
20098 [page 642]
20099 [page 643]
20100 [page 644]
20101 [page 645]
20102 [page 646]
20103 [page 647]
20104 [page 648]
20105 [page 649]
20106 [page 650]
20107 [page 651]
20108 [page 652]
20109 [page 653]
20110 [page 654]
20111 [page 655]
20112 [page 656]
20113 [page 657]
20114 [page 658]
20115 [page 659]
20116 [page 660]
20117 [page 661]
20118 [page 662]
20119 [page 663]
20120 [page 664]
20121 [page 665]
20122 [page 666]
20123 [page 667]
20124 [page 668]
20125 [page 669]
20126 [page 670]

20127 [page 671]
20128 [page 672]
20129 [page 673]
20130 [page 674]
20131 [page 675]
20132 [page 676]
20133 [page 677]
20134 [page 678]
20135 [page 679]
20136 [page 680]
20137 [page 681]
20138 [page 682]
20139 [page 683]
20140 [page 684]
20141 [page 685]
20142 [page 686]
20143 [page 687]
20144 [page 688]
20145 [page 689]
20146 [page 690]
20147 [page 691]
20148 [page 692]
20149 [page 693]
20150 [page 694]
20151 [page 695]
20152 [page 696]
20153 [page 697]
20154 [page 698]
20155 [page 699]
20156 [page 700]
20157 [page 701]
20158 [page 702]
20159 [page 703]
20160 [page 704]
20161 [page 705]
20162 [page 706]
20163 [page 707]
20164 [page 708]
20165 [page 709]
20166 [page 710]
20167 [page 711]
20168 [page 712]
20169 [page 713]

20170 [page 714]
20171 [page 715]
20172 [page 716]
20173 [page 717]
20174 [page 718]
20175 [page 719]
20176 [page 720]
20177 [page 721]
20178 [page 722]
20179 [page 723]
20180 [page 724]
20181 [page 725]
20182 [page 726]
20183 [page 727]
20184 [page 728]
20185 [page 729]
20186 [page 730]
20187 [page 731]
20188 [page 732]
20189 [page 733]
20190 [page 734]
20191 [page 735]
20192 [page 736]
20193 [page 737]
20194 [page 738]
20195 [page 739]
20196 [page 740]
20197 [page 741]
20198 [page 742]
20199 [page 743]
20200 [page 744]
20201 [page 745]
20202 [page 746]
20203 [page 747]
20204 [page 748]
20205 [page 749]
20206 [page 750]
20207 [page 751]
20208 [page 752]
20209 [page 753]
20210 [page 754]
20211 [page 755]
20212 [page 756]

20213 [page 757]
20214 [page 758]
20215 [page 759]
20216 [page 760]
20217 [page 761]
20218 [page 762]
20219 [page 763]
20220 [page 764]
20221 [page 765]
20222 [page 766]
20223 [page 767]
20224 [page 768]
20225 [page 769]
20226 [page 770]
20227 [page 771]
20228 [page 772]
20229 [page 773]
20230 [page 774]
20231 [page 775]
20232 [page 776]
20233 [page 777]
20234 [page 778]
20235 [page 779]
20236 [page 780]
20237 [page 781]
20238 [page 782]
20239 [page 783]
20240 [page 784]
20241 [page 785]
20242 [page 786]
20243 [page 787]
20244 [page 788]
20245 [page 789]
20246 [page 790]
20247 [page 791]
20248 [page 792]
20249 [page 793]
20250 [page 794]
20251 [page 795]
20252 [page 796]
20253 [page 797]
20254 [page 798]
20255 [page 799]

20256 [page 800]
20257 [page 801]
20258 [page 802]
20259 [page 803]
20260 [page 804]
20261 [page 805]
20262 [page 806]
20263 [page 807]
20264 [page 808]
20265 [page 809]
20266 [page 810]
20267 [page 811]
20268 [page 812]
20269 [page 813]
20270 [page 814]
20271 [page 815]
20272 [page 816]
20273 [page 817]
20274 [page 818]
20275 [page 819]
20276 [page 820]
20277 [page 821]
20278 [page 822]
20279 [page 823]
20280 [page 824]
20282 [page 825]
20283 [page 826]
20284 [page 827]
20285 [page 828]
20286 [page 829]
20287 [page 830]
20288 [page 831]
20289 [page 832]
20290 [page 833]
20291 [page 834]
20292 [page 835]
20293 [page 836]
20294 [page 837]
20295 [page 838]
20296 [page 839]
20297 [page 840]
20298 [page 841]
20299 [page 842]

20300 [page 843]
20301 [page 844]
20302 [page 845]

6.1 20001

Read attempted while out of synchronization with the server.

Symbolic constant

SYBESYNC

Message text

```
Read attempted while out of synchronization with the server.
```

Possible Cause

Action/solution

Contact SAP Technical Support

Additional information

Obsolete

Versions

None

6.2 20002

Server connection failed.

Symbolic constant

SYBEFCON

Message text

```
Server connection failed.
```

Possible Cause

Internal I/O error

Action/solution

Contact SAP Technical Support.

Additional information

Versions

Earlier than 15.7 ESD #3.

6.3 20003

Server connection timed out.

Symbolic constant

SYBETIME

Message text

```
Server connection timed out.
```

Possible Cause

Network I/O operation timed out.

Action/solution

Contact SAP Technical Support.

Additional information

An application can increase the amount of time to wait for a server response using `dbsettime()`.

Versions

All

6.4 20004

Read from the server failed.

Symbolic constant

SYBEREAD

Message text

```
Read from the server failed.
```

Possible cause

Internal I/O error

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.5 20005

DB-LIBRARY internal error - send buffer length corrupted.

Symbolic constant

SYBEUFL

Message text

```
DB-LIBRARY internal error - send buffer length corrupted.
```

Possible Cause

Internal memory error

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.6 20006

Write to the server failed.

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

6.7 20008

Unable to open socket.

Symbolic constant

SYBESOCK

Message text

```
Unable to open socket.
```

Possible Cause

Internal I/O error

Action/solution

Contact SAP Technical Support.

Additional information

Versions

Earlier than 15.7 ESD #3.

6.8 20009

Unable to connect socket -- server is unavailable or does not exist.

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 SAP Technical Support.

Additional information

Versions

All

6.9 20010

Unable to allocate sufficient memory.

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

6.10 20011

Maximum number of DBPROCESSes already allocated.

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

6.11 20012

Server name not found in interfaces file.

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

6.12 20013

Unknown host machine name.

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

6.13 20014

Login incorrect.

Symbolic constant

SYBEPWD

Message text

```
Login incorrect.
```

Possible Cause

Incorrect username/ password.

Action/solution

Correct username/ password.

Additional information

Versions

All

6.14 20015

Could not open interfaces file.

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

6.15 20016

Unexpected end of line.

Symbolic constant

SYBEINLN

Message text

```
interfaces file: unexpected end-of-line.
```

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

6.16 20017

Unexpected EOF from the server.

Symbolic constant

SYBESEOF

Message text

```
Unexpected EOF from the server.
```

Possible Cause

Server has been shut down.

Action/solution

Verify server status.

Additional information

Versions

Earlier than 15.7 ESD #3.

6.17 20018

Check the error messages from the server.

Symbolic constant

SYBESMSG

Message text

```
General server error: Check messages from the server.
```

Possible Cause

An error occurred on the server.

Action/solution

Correct application coding.

Additional information

Versions

All

6.18 20019

Check messages from the server.

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

6.19 20020

FIXME: Data-stream processing out of sync.

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 SAP Technical Support.

Additional information

Versions

All

6.20 20021

Illegal timeout value specified.

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 `<seconds>` parameter is a positive integer value.

Additional information

Versions

All

6.21 20022

Error in sending out-of-band data to the server.

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

6.22 20023

Unknown bind type passed to DB-LIBRARY function.

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 `<vartype>`.

Action/solution

Check function documentation for valid bind types and correct application coding.

Additional information

Versions

All

6.23 20024

Attempt to bind user variable to a non-existent compute row.

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

6.24 20025

Illegal integer column length returned by the server.

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 SAP Open Server, check SAP Open Server coding.

Additional information

Versions

All

6.25 20026

Column number out of range.

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

6.26 20027

NULL parameter not allowed for this dboption.

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

6.27 20028

Encountered variable length datatype error.

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 SAP Open Server, verify coding of SAP Open Server.

Additional information

Versions

All

6.28 20029

Encountered fixed-length datatype error.

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 SAP Open Server.

Additional information

Versions

All

6.29 20030

DB-LIBRARY internal error: ALTFMT following ALTNAME has wrong id.

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 SAP Technical Support.

Additional information

Versions

All

6.30 20031

Datastream indicates that a compute column is derived from a non-existent select-list member.

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 SAP Technical Support.

Additional information

Versions

All

6.31 20033

User attempted a dbbind() with mismatched column and variable types.

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 `<vartype>` values.

Action/solution

Correct application coding.

Additional information

Versions

All

6.32 20034

Attempt to bind using NULL pointers.

Symbolic constant

SYBEABNP

Message text

```
Attempt to bind using NULL pointers.
```

Possible Cause

Bad `<varaddr>` argument to `dbbind()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.33 20035

User attempted a dbbind() with mismatched column and variable types.

Symbolic constant

SYBEAAMT

Message text

```
User attempted a dbbind() with mismatched column and variable types.
```

Possible Cause

Program datetype does not match column datetype.

Action/solution

Correct application coding.

Additional information

Versions

All

6.34 20036

The server did not grant us a distributed-transaction ID.

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

6.35 20037

The server did not recognize our distributed-transaction ID.

Symbolic constant

SYBERXID

Message text

```
The server did not recognize our distributed-transaction ID.
```

Possible Cause

Programming error.

Action/solution

Validate `<commid>` parameter specified to `stat_xact()`.

Additional information

Versions

All

6.36 20038

Invalid computeid or compute column number.

Symbolic constant

SYBEICN

Message text

```
Invalid computeid or compute column number.
```

Possible Cause

Invalid `<computeid>` or compute column number passed to a `dbalt*()` routine.

Action/solution

Correct application coding.

Additional information

Versions

All

6.37 20039

No such member of 'order by' clause.

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

6.38 20040

Attempt to print unknown token.

Symbolic constant

SYBEAPUT

Message text

```
Attempt to print unknown token.
```

Possible Cause

Unknown `<type>` parameter passed to `dbprtype()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.39 20041

Attempt to set fields in a null loginrec.

Symbolic constant

SYBEASNL

Message text

```
Attempt to set fields in a null loginrec.
```

Possible Cause

`LOGINREC` 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

6.40 20042

Name too long for loginrec field.

Symbolic constant

SYBENTLL

Message text

```
Name too long for loginrec field.
```

Possible Cause

`<name>` parameter to `DBSETL*` routine is longer than `<DBMAXNAME>`.

Action/solution

Use a shorter name.

Additional information

Versions

All

6.41 20043

Attempt to set unknown loginrec field.

loginrec==>loginrec

Symbolic constant

SYBEASUL

Message text

```
Attempt to set unknown loginrec field.
```

Possible Cause

Internal function `dbsetlname` attempted to set a nonexistent `<loingrec>` field.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.42 20044

Attempt to retrieve data from a non-existent row.

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

6.43 20045

Negative starting index passed to dbstrcpy().

Symbolic constant

SYBENSI

Message text

```
Negative starting index passed to dbstrcpy().
```

Possible Cause

`<start>` parameter to `dbstrcpy()` is less than zero.

Action/solution

Correct value.

Additional information

Versions

All

6.44 20046

Attempt to bind to a NULL program variable.

Symbolic constant

SYBEABNV

Message text

```
Attempt to bind to a NULL program variable.
```

Possible Cause

`<destvar>` parameter to dbbind is NULL.

Action/solution

Provide a pointer to memory for the program variable.

Additional information

Versions

All

6.45 20047

DBPROCESS is dead or not enabled.

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 SAP Technical Support.

Additional information

Versions

All

6.46 20048

Data-conversion resulted in underflow.

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

6.47 20049

Data-conversion resulted in underflow.

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

6.48 20050

Attempt to convert data stopped by syntax error in source field.

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

6.49 20051

Data-conversion resulted in loss of precision.

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

6.50 20052

Attempt to set variable to NULL resulted in overflow.

Symbolic constant

SYBECNOV

Message text

```
Attempt to set variable to NULL resulted in overflow.
```

Possible Cause

`dbconvert()` `<src>` parameter should not be NULL.

Action/solution

Correct application coding.

Additional information

Versions

All

6.51 20053

Requested data-conversion does not exist.

Symbolic constant

SYBERDCN

Message text

```
Requested data-conversion does not exist.
```

Possible Cause

`dbconvert()` is unable to convert from `<srctype>` to `<desttype>`.

Action/solution

Check the datatype conversion table for valid type pairs.

Additional information

Versions

All

6.52 20054

dbsafestr() overflowed its destination buffer.

Symbolic constant

SYBESFOV

Message text

```
dbsafestr() overflowed its destination buffer.
```

Possible Cause

Destination buffer is not large enough.

Action/solution

Correct application coding.

Additional information

Versions

All

6.53 20055

Unknown network type found in interfaces file.

Symbolic constant

SYBEUNT

Message text

```
Unknown network type found in interfaces file.
```

Possible Cause

Interfaces file has an erroneous server entry.

Action/solution

Correct interfaces file.

Additional information

Versions

Earlier than 15.7 ESD #3.

6.54 20056

Error in closing network connection.

Symbolic constant

SYBECLOS

Message text

```
Error in closing network connection.
```

Possible Cause

There was an error closing the network endpoint.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.55 20060

Unknown datatype encountered.

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

6.56 20061

Attempt to call dbtsput() with an invalid timestamp.

Symbolic constant

SYBETSIT

Message text

```
Attempt to call dbtsput() with an invalid timestamp.
```

Possible Cause

Adaptive Server column has no timestamp.

Action/solution

Correct application coding.

Additional information

Versions

All

6.57 20062

Attempt to update the timestamp of a table which has no timestamp column.

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

6.58 20063

Bad bulk-copy direction. Must be either IN or OUT.

Symbolic constant

SYBEBDIO

Message text

```
Bad bulk-copy direction. Must be either IN or OUT.
```

Possible Cause

`bcp_init()` called with invalid `<direction>` parameter.

Action/solution

Correct application coding.

Additional information

Versions

All

6.59 20064

Attempt to use Bulk Copy with a non-existent server table.

Symbolic constant

SYBEBCNT

Message text

```
Attempt to use Bulk Copy with a non-existent server table.
```

Possible Cause

`<table>` parameter to `bcp_init()` is invalid for server.

Action/solution

Correct application coding.

Additional information

Versions

All

6.60 20065

Illegal field number passed to bcp_control().

Symbolic constant

SYBEIFNB

Message text

```
Illegal field number passed to bcp_control().
```

Possible Cause

`<field>` parameter to `bcp_control()` is incorrect.

Action/solution

Correct application coding.

Additional information

Versions

All

6.61 20066

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.

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

6.62 20067

1000 rows successfully bulk-copied to host-fi

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

6.63 20068

Batch successfully bulk-copied to the server.

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

6.64 20069

Bcp:1000 rows sent to the server.

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

6.65 20070

I/O error while reading bcp data-file.

Symbolic constant

SYBEBCRE

Message text

```
I/O error while reading bcp data-file.
```

Possible Cause

System I/O routing failed.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.66 20071

Syntax error:only two periods are permitted in table names.

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

6.67 20072

I/O error while writing bcp data-file.

Symbolic constant

SYBEBCWE

Message text

```
I/O error while writing bcp data-file.
```

Possible Cause

System I/O routing failed.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.68 20073

Attempt to bulk-copy a NULL value into server column <colname>, which does not accept NULL values.

Symbolic constant

SYBEBCNN

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

6.69 20074

Attempt to bulk-copy an oversized row to the server.

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 `bcp` file/server schema.

Additional information

Versions

All

6.70 20075

Attempt to bulk-copy an illegally-sized column value to the server.

Symbolic constant

SYBEBCIS

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 `bcp` file/server schema.

Additional information

Versions

All

6.71 20076

bcp_init() must be called before any other bcp routines.

Symbolic constant

SYBEBCPI

Message text

```
bcp_init() must be called before any other bcp routines.
```

Possible Cause

`bcp_init()` not called.

Action/solution

Call `bcp_init()`.

Additional information

Versions

All

6.72 20077

bcp_bind(), bcp_collen() and bcp_colptr() may be used only after bcp_init() has been called with the copy direction set to DB_IN.

Symbolic constant

SYBEBCOR

Message text

```
bcp_bind(), bcp_collen() and bcp_colptr() may be used only after bcp_init() has
been called with the copy direction set to DB_IN.
```

Possible Cause

Above functions called before `bcp_init()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.73 20078

bcp_bind() may NOT be used after bcp_init() has been passed a non-NULL input file name.

Symbolic constant

SYBEBCPB

Message text

```
bcp_bind() may NOT be used after bcp_init() 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

6.74 20079

For bulk copy, all variable-length data must have either a length-prefix or a terminator specified.

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 `<length-prefix>` or `<terminator>`.

Action/solution

Correct parameters to `bcp_colfmt()`.

Additional information

Versions

All

6.75   20080

bcp_columns() and bcp_colfmt() may be used only after bcp_init() has been passed a valid input file.

Symbolic constant

SYBEBIVI

Message text

```
bcp_columns() and bcp_colfmt() may be used only after bcp_init() has been passed
a valid input file.
```

Possible Cause

`bcp_init()` has been called with an invalid `<hfile>` parameter.

Action/solution

Correct application coding.

Additional information

Versions

All

6.76 20081

bcp_columns() must be called before bcp_colfmt().

Symbolic constant

SYBEBCBC

Message text

```
bcp_columns() must be called before bcp_colfmt().
```

Possible Cause

`bcp_colfmt()` has been called before calling `bcp_columns()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.77 20082

Bcp host-files must contain at least one column.

Symbolic constant

SYBEBCFO

Message text

```
Bcp host-files must contain at least one column.
```

Possible Cause

`bcp_columns()` has been called with an invalid `<host_colcount>` parameter.

Action/solution

Correct application coding.

Additional information

Versions

All

6.78 20083

bcp_exec() may be called only after bcp_init() has been passed a valid host file.

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

6.79 20084

Bcp: Unable to open host data-file.

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

6.80 20085

Bcp: Unable to close host data-file.

Symbolic constant

SYBEBCUC

Message text

```
Bcp: Unable to close host data-file.
```

Possible Cause

System close routine failed.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.81 20086

Bcp: Unable to open error-file.

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

6.82 20087

Bcp: Unable to open error-file.

open==>close

Symbolic constant

SYBEBUCE

Message text

```
Bcp: Unable to open error-file.
```

Possible Cause

System close routine failed.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.83 20088

I/O error while writing bcp error-file.

Symbolic constant

SYBEBWEF

Message text

```
I/O error while writing bcp error-file.
```

Possible Cause

System I/O routine failed.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.84 20091

Attempt to send an empty command buffer to the server.

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

6.85 20092

Attempt to send too much TEXT data via the dbmoretext() call.

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

6.86 20093

Attempt to use dbtxtsput() to put a new text-timestamp into a non-existent data row.

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

6.87 20094

Attempt to use dbtxtsput() to put a new text-timestamp into a column whose datatype is neither SYBTEXT nor SYBIMAGE.

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

6.88 20095

Attempt to send too much TEXT data via the bcp_moretext() call.

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

6.89 20096

Attempt to set remote password would overflow the login-record's remote-password field.

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

6.90 20097

Attempt to read an unknown version of BCP format-file.

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 SAP Technical Support.

Additional information

Versions

All

6.91 20098

Bcp: Unable to open format-file.

Symbolic constant

SYBEBUOF

Message text

```
Bcp: Unable to open format-file.
```

Possible Cause

Incorrect `filename` parameter passed to `bcp_readfmt()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.92 20099

Bcp: Unable to close format-file.

Symbolic constant

SYBEBUCF

Message text

```
Bcp: Unable to close format-file.
```

Possible Cause

System `close` routine failed.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.93 20100

I/O error while reading bcp format-file.

Symbolic constant

SYBEBRFF

Message text

```
I/O error while reading bcp format-file.
```

Possible Cause

System I/O routine failed.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.94 20101

I/O error while writing bcp format-file.

Symbolic constant

SYBEBWFF

Message text

```
I/O error while writing bcp format-file.
```

Possible Cause

System I/O routine failed.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.95 20102

Bcp: Unrecognized datatype found in format-file.

Symbolic constant

SYBEBUDF

Message text

```
Bcp: Unrecognized datatype found in format-file.
```

Possible Cause

Corrupted input file.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.96 20103

Incorrect host-column number found in bcp format-file.

Symbolic constant

SYBEBIHC

Message text

```
Incorrect host-column number found in bcp format-file.
```

Possible Cause

Corrupted input file.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.97 20104

Unexpected EOF encountered in BCP data-file.

Symbolic constant

SYBEBEOF

Message text

```
Unexpected EOF encountered in BCP data-file.
```

Possible Cause

Corrupted input file.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.98 20105

Negative length-prefix found in BCP data-file.

Symbolic constant

SYBEBCNL

Message text

```
Negative length-prefix found in BCP data-file.
```

Possible Cause

Corrupted input file.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.99 20106

Host-file columns may be skipped only when copying Into the server.

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

6.100 20107

It's illegal to use BCP terminators with program variables other than SYBCHAR, SYBBINARY, SYBTEXT, or SYBIMAGE.

Symbolic constant

SYBEBCIT

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

6.101 20108

The BCP hostfile <filename> contains only <n> rows. Skipping all of these rows is not allowed.

Symbolic constant

SYBEBCSA

Message text

```
The BCP hostfile <filename> contains only <n> rows. Skipping all of these rows
is not allowed.
```

Possible Cause

`bcp_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

6.102 20109

NULL DBPROCESS pointer passed to DB-Library.

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

6.103 20110

Unable to get current username from operating system.

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

6.104 20111

The BCP hostfile <filename> contains only <n> rows. It was impossible to read the requested <m> rows.

Symbolic constant

SYBEBCRO

Message text

```
The BCP hostfile <filename> contains only <n> rows. It was impossible to read
the requested <m> rows.
```

Possible Cause

`bcp_control()` has set `<BCPLAST>` to a value greater than the number of rows in the input file.

Action/solution

Correct application coding.

Additional information

Versions

All

6.105 20112

Attempt to set maximum number of DBPROCESSes lower than 1.

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

6.106 20113

It is illegal to pass -1 to dbrpcparam() for the datalen of parameters which are of type SYBCHAR, SYBVARCHAR, SYBBINARY, or SYBVARBINARY.

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 datetype.

Action/solution

Correct application coding.

Additional information

Versions

All

6.107 20114

When passing a SYBINTN, SYBDATETIMN, SYBMONEYN, or SYBFLTN 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.

Symbolic constant

SYBERPUL

Message text

```
When passing a SYBINTN, SYBDATETIMN, SYBMONEYN, or SYBFLTN 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

6.108 20115

Unknown option passed to dbsetopt().

Symbolic constant

SYBEUNOP

Message text

```
Unknown option passed to dbsetopt().
```

Possible Cause

`dbsetopt()` called incorrectly.

Action/solution

Correct application coding.

Additional information

Versions

All

6.109 20116

The current row is not a result of compute clause <computeid>, so it is illegal to attempt to extract that data from this row.

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 data from this row.
```

Possible Cause

`dbadata()` 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

6.110 20117

dbreadtext() may not be used to receive the results of a query which contains a COMPUTE clause.

Symbolic constant

SYBERTCC

Message text

```
dbreadtext() may not be used to receive the results of a query which contains a
COMPUTE clause.
```

Possible Cause

`dbreadtext()` has been called to retrieve text from the result of an alter row clause.

Action/solution

Correct application coding.

Additional information

Versions

All

6.111 20118

dbreadtext() may only be used to receive the results of a query which contains a single result column.

Symbolic constant

SYBERTSC

Message text

```
dbreadtext() may only be used to receive the results of a query which contains a
single result column.
```

Possible Cause

`dbreadtext()` has been called on a result set containing more than one column.

Action/solution

Correct application coding.

Additional information

Versions

All

6.112 20119

Internal software error: Unknown connection result reported by dbpasswd().

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 SAP Technical Support.

Additional information

Versions

All

6.113 20120

The RPC facility is available only when using a server whose version number is 4.0 or greater.

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

6.114 20121

The text/image facility is available only when using a server whose version number is 4.0 or greater.

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

6.115 20122

Bcp: Row number of the first row to be copied cannot be greater than the row number for the last row to be copied.

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 `<BCPFIRST>` and `<BCPLAST>`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.116 20123

Attempt to set column width less than 1.

Symbolic constant

SYBECWLL

Message text

```
Attempt to set column width less than 1.
```

Possible Cause

Action/solution

Contact SAP Technical Support.

Additional information

Obsolete

Versions

None

6.117 20124

Unrecognized format encountered in dbstrbuild().

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

6.118 20125

Unrecognized custom-format parameter-type encountered in dbstrbuild().

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

6.119 20126

Attempt to install too many custom formats via (obsolete function) dbfmtinstall().

Symbolic constant

SYBETMCF

Message text

```
Attempt to install too many custom formats via (obsolete function)
dbfmtinstall().
```

Possible Cause

`dbfmtinstall()` called with more than `<MAXFMTS (20)>` formats.

Action/solution

Correct application coding.

Additional information

Versions

All

6.120 20127

Error in attempting to install custom format.

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

6.121 20128

Error in attempting to determine the size of a pair of translation tables.

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 `<srv_charset>`.

Action/solution

Correct application coding.

Additional information

Make sure specified character set exists in `$SYBASE/charsets`.

Versions

All

6.122 20129

Error in attempting to load a pair of translation tables.

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

6.123 20130

Error in attempting to perform a character-set translation.

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

6.124 20131

A character-set translation overflowed its destination buffer while using bcp to copy data from a host-file to the server.

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 SAP Technical Support.

Additional information

Versions

All

6.125 20132

Error in attempting to find the Sybase home directory.

Symbolic constant

SYBEFSHD

Message text

```
Error in attempting to find the Sybase home directory.
```

Possible Cause

`$SYBASE` environment variable is incorrect.

Action/solution

Correct setting.

Additional information

Versions

All

6.126 20133

Error in attempting to open a localization file.

Symbolic constant

Message text

```
Error in attempting to open a localization file.
```

Possible Cause

Unable to open a localization file.

Action/solution

Make sure `$SYBASE` is correct.

Additional information

Versions

All

6.127 20134

Error in attempting to read datetime information from a localization file.

Symbolic constant

SYBEARDI

Message text

```
Error in attempting to read datetime information from a localization file.
```

Possible Cause

Corrupt `locales` file.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.128 20135

Unable to read copyright information\from the dblib localization file.

Symbolic constant

SYBEURCI

Message text

```
Unable to read copyright information\from the dblib localization file.
```

Possible Cause

Corrupt `locales` file.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.129 20136

Error in attempting to read the dblib.loc localization file.

Symbolic constant

SYBEARDL

Message text

```
Error in attempting to read the dblib.loc localization file.
```

Possible Cause

Corrupt `locales` file.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.130 20137

Unable to read money-format information from the dblib localization file.

Symbolic constant

SYBEURMI

Message text

```
Unable to read money-format information from the dblib localization file.
```

Possible Cause

Corrupt `locales` file.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.131 20138

Unable to read error mnemonic from the dblib localization file.

Symbolic constant

SYBEUREM

Message text

```
Unable to read error mnemonic from the dblib localization file.
```

Possible Cause

Corrupt `locales` file.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.132 20139

Unable to read error string from the dblib localization file.

Symbolic constant

SYBEURES

Message text

```
Unable to read error string from the dblib localization file.
```

Possible Cause

Corrupt `locales` file.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.133 20140

Unable to read error information from the dblib localization file.

Symbolic constant

SYBEUREI

Message text

```
Unable to read error information from the dblib localization file.
```

Possible Cause

Corrupt `locales` file.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.134 20141

Warning: an out-of-range error-number was encountered in dblib.loc. The maximum permissible error-number is defined as DBERRCOUNT in sybdb.h.

Symbolic constant

SYBEOREN

Message text

```
Warning: an out-of-range error-number was encountered in dblib.loc. The maximum
permissible error-number is defined as DBERRCOUNT in sybdb.h.
```

Possible Cause

Corrupt `locales` file.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.135 20142

Invalid sort-order information found.

Symbolic constant

SYBEISOI

Message text

```
Invalid sort-order information found.
```

Possible Cause

Corrupt `locales` file.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.136 20143

Illegal datetime column length returned by the server. Legal datetime lengths are 4 and 8 bytes.

Symbolic constant

SYBEIDCL

Message text

```
Illegal datetime column length returned by the server. Legal datetime lengths
are 4 and 8 bytes.
```

Possible Cause

Internal Adaptive Server error.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.137 20144

Illegal money column length returned by the server. Legal money lengths are 4 and 8 bytes.

Symbolic constant

SYBEIMCL

Message text

```
Illegal money column length returned by the server. Legal money lengths are 4
and 8 bytes.
```

Possible Cause

Internal Adaptive Server error.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.138 20145

Illegal floating-point column length returned by the server. Legal floating-point lengths are 4 and 8 bytes.

Symbolic constant

SYBEIFCL

Message text

```
Illegal floating-point column length returned by the server. Legal floating-
point lengths are 4 and 8 bytes.
```

Possible Cause

Internal Adaptive Server error.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.139 20146

Unrecognized TDS version received from the server.

Symbolic constant

SYBEUTDS

Message text

```
Unrecognized TDS version received from the server.
```

Possible Cause

Internal error.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.140 20147

Bcp: Unable to create format-file.

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

6.141 20148

Attempt to do data-conversion with NULL destination variable.

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

6.142 20149

Out-of-range datepart constant.

Symbolic constant

SYBEDPOR

Message text

```
Out-of-range datepart constant.
```

Possible Cause

Invalid `<datepart>` passed to `dbdatechar()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.143 20150

Cannot have negative component in the date in numeric form.

Symbolic constant

SYBENDC

Message text

```
Cannot have negative component in the date in numeric form.
```

Possible Cause

Invalid `<value>` passed to `dbdatechar()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.144 20151

Month values must be between 1 and 12.

Symbolic constant

SYBEMVOR

Message text

```
Month values must be between 1 and 12.
```

Possible Cause

Invalid `<month>` value passed to `dbdatechar()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.145 20152

Day values must be between 1 and 7.

Symbolic constant

SYBEDVOR

Message text

```
Day values must be between 1 and 7.
```

Possible Cause

Invalid `<day>` value passed to `dbdatechar()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.146 20153

Cannot pass dbsetnull()a NULL bindval pointer.

Symbolic constant

SYBENBVP

Message text

```
Cannot pass dbsetnull()a NULL bindval pointer.
```

Possible Cause

Null pointer passed as `<bindval>`.

Action/solution

Correct application coding.

Additional information

When a null value is encountered, bindval is the value to use instead. NULL is inappropriate here.

Versions

All

6.147 20154

Called dbspid() with a NULL dbproc.

Symbolic constant

SYBESPID

Message text

```
Called dbspid() with a NULL dbproc.
```

Possible Cause

Incorrect coding.

Action/solution

Correct application coding.

Additional information

Versions

All

6.148 20155

Called dbdatecrack() with NULL datetime parameter.

Symbolic constant

SYBENDTP

Message text

```
Called dbdatecrack() with NULL datetime parameter.
```

Possible Cause

Incorrect coding.

Action/solution

Correct application coding.

Additional information

Versions

All

6.149 20156

The xlt_tosrv and xlt_todisp parameters to dbfree_xlate() were NULL.

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

6.150 20157

Warning: the xlt_todisp parameter to dbfree_xlate() was NULL. The space associated with the xlt_tosrv parameter has been freed.

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 `<xlt_todisp>` parameter passed to `dbfree_xlate()`.

Action/solution

Do not free `<xlt_tosr>`v.

Additional information

Versions

All

6.151 20158

Warning: the xlt_tosrv parameter to dbfree_xlate() was NULL. The space associated with the xlt_todisp parameter has been freed.

Symbolic constant

SYBEXTSN

Message text

```
Warning: the xlt_tosrv parameter to dbfree_xlate() was NULL. The space
associated with the xlt_todisp parameter has been freed.
```

Possible Cause

Null `<xlt_tosrv>` parameter passed to `dbfree_xlate()`.

Action/solution

Do not free `<xlt_todisp>`.

Additional information

Versions

All

6.152 20159

Incorrect number of arguments given to DB-Library.

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

6.153 20160

Invalid argument type given to Hyper/DB-Library.

Symbolic constant

SYBETYPE

Message text

```
Invalid argument type given to Hyper/DB-Library.
```

Possible Cause

`dbregparam()` called with an invalid type.

Action/solution

Correct application coding.

Additional information

Versions

All

6.154 20161

General operating system error.

Symbolic constant

SYBEGENOS

Message text

```
General operating system error.
```

Possible Cause

Creating an internal condition variable failed.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.155 20162

Wrong resource type or length given for dbpage??() operation.

Symbolic constant

SYBEPAGE

Message text

```
Wrong resource type or length given for dbpage??() operation.
```

Possible Cause

Action/solution

Contact SAP Technical Support.

Additional information

Obsolete.

Versions

None

6.156 20163

Option not allowed.

Symbolic constant

SYBEOPTNO

Message text

```
Option not allowed.
```

Possible Cause

Internal failure to set security option.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.157 20164

Failure to send the expected amount of TEXT or IMAGE data via dbmoretext().

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

6.158 20165

Invalid resource type given to Hyper/DB-Library.

Symbolic constant

SYBERTYPE

Message text

```
Invalid resource type given to Hyper/DB-Library.
```

Possible Cause

Action/solution

Contact SAP Technical Support.

Additional information

Obsolete

Versions

None

6.159 20166

Cannot open resource file.

Symbolic constant

SYBERFILE

Message text

```
Cannot open resource file.
```

Possible Cause

Action/solution

Contact SAP Technical Support.

Additional information

Obsolete

Versions

None

6.160 20167

Read/Write/Append mode denied on file.

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

6.161 20168

Could not select or copy field specified.

Symbolic constant

SYBESLCT

Message text

```
Could not select or copy field specified.
```

Possible Cause

Action/solution

Contact SAP Technical Support.

Additional information

Obsolete

Versions

None

6.162 20169

Attempt to send zero length TEXT or IMAGE to dataserver via dbwritetext().

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

6.163 20170

VMS: The file being opened must be a stream_lf.

Message type

Error

Symbolic constant

SYBENTST

Message text

```
VMS: The file being opened must be a stream_lf.
```

Possible Cause

Action/solution

Contact SAP Technical Support.

Additional information

Obsolete

Versions

None

6.164 20171

Login incorrect: operating system login security level is not within the range of the secure server.

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 SAP Technical Support.

Additional information

Obsolete

Versions

None

6.165 20172

Login incorrect: login security level as entered does not agree with your operating system level.

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 SAP Technical Support.

Additional information

Obsolete

Versions

None

6.166 20173

Program not linked with specified network library.

Symbolic constant

SYBENLNL

Message text

```
Program not linked with specified network library.
```

Possible Cause

Action/solution

Contact SAP Technical Support.

Additional information

Obsolete

Versions

None

6.167 20174

Called dbrecvpassthru() with a NULL handle parameter.

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

6.168 20175

Called dbsendpassthru() with a NULL buf parameter.

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

6.169 20176

Called <routine> with a NULL <paramname> parameter.

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

6.170 20177

An event handler must be installed before a notification request can be made.

Symbolic constant

SYBENOTI

Message text

```
An event handler must be installed before a notification request can be made.
```

Possible Cause

Action/solution

Contact SAP Technical Support.

Additional information

Obsolete

Versions

None

6.171 20178

Called dbregwatch() with a bad options parameter.

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

6.172 20179

Called dbreghandle() with a NULL handler parameter.

Symbolic constant

SYBENEHA

Message text

```
Called dbreghandle() with a NULL handler parameter.
```

Possible Cause

Action/solution

Contact SAP Technical Support.

Additional information

Obsolete

Versions

None

6.173 20180

DBPROCESS is being used for another transaction.

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

6.174 20181

Must initiate a transaction before calling dbregparam().

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

6.175 20182

Must call dbreginit() before dbregexec().

Symbolic constant

SYBEEINI

Message text

```
Must call dbreginit() before dbregexec().
```

Possible Cause

`dbreginit()` has not been called before the invocation of `dbregexec()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.176 20183

Must call dbnpdefine() before dbnpcreate().

Symbolic constant

SYBEECRT

Message text

```
Must call dbnpdefine() before dbnpcreate().
```

Possible Cause

Action/solution

Contact SAP Technical Support.

Additional information

Obsolete

Versions

None

6.177 20184

Attempted to cancel unrequested event notification.

Symbolic constant

SYBEECAN

Message text

```
Attempted to cancel unrequested event notification.
```

Possible Cause

`dbregnowatch()` has been called with no prior `dbregwatch()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.178 20185

Unsolicited event notification received.

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

6.179 20186

Must call dbrpcinit() before dbrpcparam() or dbrpcsend().

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

6.180 20187

No SYBTEXT or SYBIMAGE parameters were defined.

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

6.181 20188

Called dbmoretext() with a bad size parameter.

Symbolic constant

SYBETEXS

Message text

```
Called dbmoretext() with a bad size parameter.
```

Possible Cause

`dbmoretext()` or `dbreadtext()` called with a negative `<bufsize>` parameter.

Action/solution

Correct application coding.

Additional information

Versions

All

6.182 20189

Attempted to turn off a trace flag that was not on.

Symbolic constant

SYBETRAC

Message text

```
Attempted to turn off a trace flag that was not on.
```

Possible Cause

`dbtraceoff()` called with an invalid `<flag>` parameter.

Action/solution

Correct application coding.

Additional information

Versions

All

6.183 20190

DB-Library internal error - trace structure not found.

Symbolic constant

SYBETRAS

Message text

```
DB-Library internal error - trace structure not found.
```

Possible Cause

There is no trace record in the DBPROCESS structure.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.184 20191

dbtracestring() may only be called from a printfunc().

Symbolic constant

SYBEPRTF

Message text

```
dbtracestring() may only be called from a printfunc().
```

Possible Cause

`dbtracestring()` called directly instead of from `printfunc()` set with `dbtraceon()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.185 20192

Bad numbytes parameter passed to dbtracestring().

Symbolic constant

SYBETRSN

Message text

```
Bad numbytes parameter passed to dbtracestring().
```

Possible Cause

The `<num>` parameter passed to `dbtracestring()` is negative.

Action/solution

Correct application coding.

Additional information

Versions

All

6.186 20193

In DBSETLPACKET(), the packet size parameter must be between 256 and 9999.

Symbolic constant

SYBEBPKS

Message text

```
In DBSETLPACKET(), the packet size parameter must be between 256 and 9999.
```

Possible Cause

Invalid size passed to `DBSETLPACKET()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.187 20194

<value> is an illegal value for the <paramname> parameter of <routine>.

Symbolic constant

SYBEIPV

Message text

```
<value> is an illegal value for the <paramname> parameter of <routine>.
```

Possible Cause

Parameter value is outside the domain of the parameter.

Action/solution

Correct application coding.

Additional information

Versions

All

6.188 20195

Money arithmetic resulted in overflow in function <routine>.

Symbolic constant

SYBEMOV

Message text

```
Money arithmetic resulted in overflow in function <routine>.
```

Possible Cause

Invalid parameter passed to a `dbmny*()` function.

Action/solution

Correct application coding.

Additional information

Versions

All

6.189 20196

Attempt to divide by $0.00 in function <routine>.

Symbolic constant

SYBEDIVZ

Message text

```
Attempt to divide by $0.00 in function <routine>.
```

Possible Cause

Invalid value passed to `dbmydiv()` or `dbmny4div()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.190 20197

Synchronous I/O attempted at AST level.

Symbolic constant

SYBEASTL

Message text

```
Synchronous I/O attempted at AST level.
```

Possible Cause

Action/solution

Contact SAP Technical Support.

Additional information

Obsolete VMS-specific error message

Versions

None

6.191 20198

DB_SETEVENT_VMS cannot be called if connections are present.

Symbolic constant

SYBESEFA

Message text

```
DB_SETEVENT_VMS cannot be called if  connections are present.
```

Possible Cause

Action/solution

Contact SAP Technical Support.

Additional information

Obsolete VMS-specific error message

Versions

None

6.192 20199

There is already an active dbpoll().

Symbolic constant

SYBEPOLL

Message text

```
There is already an active dbpoll().
```

Possible Cause

Action/solution

Contact SAP Technical Support.

Additional information

Obsolete VMS-specific error message

Versions

None

6.193 20200

DBPOLL cannot be called when registered procedure notifications have been disabled.

Symbolic constant

SYBENOEV

Message text

```
DBPOLL cannot be called when registered procedure notifications have been
disabled.
```

Possible Cause

Action/solution

Contact SAP Technical Support.

Additional information

Obsolete VMS-specific error message

Versions

None

6.194 20201

Packet size of <requested size> not supported -- size of <other size> used instead.

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

6.195 20202

Secure server function not supported in this version.

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

6.196 20203

DB-Library capabilities not accepted by the server.

Symbolic constant

SYBECAP

Message text

```
DB-Library capabilities not accepted by the server.
```

Possible Cause

Invalid TDS received from server.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.197 20204

Functionality not supported at the specified version level.

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

6.198 20205

Response function address passed to dbresponse() must be non-NULL.

Symbolic constant

SYBERESP

Message text

```
Response function address passed to dbresponse() must be non-NULL.
```

Possible Cause

The `<response_fun>`c parameter passed to undocumented function `dbresponse()` is NULL.

Action/solution

Correct application coding.

Additional information

Versions

All

6.199 20206

Illegal version level specified.

Symbolic constant

SYBEIVERS

Message text

```
Illegal version level specified.
```

Possible Cause

Invalid `<version>` parameter passed to `dbsetversion()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.200 20207

Function can be called only once.

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

6.201 20208

value parameter for dbprcparam() can be NULL, only if the datalen parameter is 0.

Symbolic constant

SYBERPNULL

Message text

```
value parameter for dbprcparam() can be NULL, only if the datalen parameter is 0.
```

Possible Cause

Parameters to `dbrpcparam()` are not in agreement.

Action/solution

Correct application coding.

Additional information

Versions

All

6.202 20209

RPC parameters cannot be of type Text/Image.

Symbolic constant

SYBERPTXTIM

Message text

```
RPC parameters cannot be of type Text/Image.
```

Possible Cause

The `<type>` parameter passed to `dbrpcparam()` cannot be be SYBTEXT or SYBIMAGE.

Action/solution

Correct application coding.

Additional information

Versions

All

6.203 20210

Negotiated login attempt failed.

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

6.204 20211

Security labels should be less than 256 characters long.

Symbolic constant

SYBELBLEN

Message text

```
Security labels should be less than 256 characters long.
```

Possible Cause

Label values passed to undocumented routine `dbsetsecurity()` exceed `<DB_MAX_LABELLEN>`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.205 20212

Unknown message-id in MSG datastream.

Symbolic constant

SYBEUMSG

Message text

```
Unknown message-id in MSG datastream.
```

Possible Cause

Internal TDS error.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.206 20213

Unexpected capability type in CAPABILITY datastream.

Symbolic constant

SYBECAPTYP

Message text

```
Unexpected capability type in CAPABILITY datastream.
```

Possible Cause

Internal TDS error.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.207 20214

Bad numbytes parameter passed to dbstrcpy().

Symbolic constant

SYBEBNUM

Message text

```
Bad numbytes parameter passed to dbstrcpy().
```

Possible Cause

An invalid value for the `<numbytes>` parameter has been passed to the `dbstrcpy()` routine.

Action/solution

Correct application coding.

Additional information

Versions

All

6.208 20215

Bad bindlen parameter passed to dbsetnull().

Symbolic constant

SYBEBBL

Message text

```
Bad bindlen parameter passed to dbsetnull().
```

Possible Cause

A negative value has been passed in `<bindlen>` parameter of the DB-Library routine `dbsetnull()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.209 20216

Illegal precision specified.

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

6.210 20217

Illegal scale specified.

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

6.211 20218

Source field value is not within the domain of legal values.

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

6.212 20219

Internal Conversion error.

Symbolic constant

SYBECINTERNAL

Message text

```
Internal Conversion error.
```

Possible Cause

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.213 20220

Datatype is not supported by the server.

Symbolic constant

SYBEBTYPSRV

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

6.214 20221

Unknown character-set encountered.

Symbolic constant

SYBEBCSET

Message text

```
Unknown character-set encountered.
```

Possible Cause

Server specified an unrecognized character set.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.215 20222

Password Encryption failed.

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 SAP Technical Support.

Additional information

Versions

All

6.216 20223

Challenge-Response function failed.

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 SAP Technical Support.

Additional information

Versions

All

6.217 20224

Illegal precision returned by the server.

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 SAP Technical Support.

Additional information

Versions

All

6.218 20225

Illegal scale returned by the server.

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 SAP Technical Support.

Additional information

Versions

All

6.219 20226

Invalid numeric column length returned by the server.

Symbolic constant

SYBEINUMCL

Message text

```
Invalid numeric column length returned by the server.
```

Possible Cause

Illegal value sent by the server.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.220 20227

Invalid decimal column length returned by the server.

Symbolic constant

SYBEIDECCL

Message text

```
Invalid decimal column length returned by the server.
```

Possible Cause

Illegal value sent by the server.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.221 20228

bcp_moretext() may be used only when there is at least one text or image column in the server table.

Symbolic constant

SYBEBCMTXT

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

6.222 20229

Column <column>: Illegal precision value encountered.

Symbolic constant

SYBEBCPREC

Message text

```
Column <column>: Illegal precision value encountered.
```

Possible Cause

Invalid precision value found in the host file.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.223 20230

bcp_bind(): if varaddr is NULL, prefixlen must be 0 and no terminator should be specified.

Symbolic constant

SYBEBCBNPR

Message text

```
bcp_bind(): if varaddr is NULL, prefixlen must be 0 and no terminator should be
specified.
```

Possible Cause

`bcp_bind()` called incorrectly.

Action/solution

Correct application coding.

Additional information

Versions

All

6.224 20231

bcp_bind(): if varaddr is NULL and varlen greater than 0, the table column type must be SYBTEXT or SYBIMAGE and the program variable type must be SYBTEXT, SYBCHAR, SYBIMAGE or SYBBINARY.

Symbolic constant

SYBEBCBNTYP

Message text

```
bcp_bind(): if varaddr is NULL and varlen greater than 0, the table column type
must be SYBTEXT or SYBIMAGE and the program variable type must be SYBTEXT,
SYBCHAR, SYBIMAGE or SYBBINARY.
```

Possible Cause

`bcp_bind()` called incorrectly.

Action/solution

Correct application coding.

Additional information

Versions

All

6.225 20232

column number <colnum>: if varaddr is NULL and varlen greater than 0, the table column type must be SYBTEXT or SYBIMAGE and the program variable type must be SYBTEXT, SYBCHAR, SYBIMAGE or SYBBINARY.

Symbolic constant

SYBEBCSNTYP

Message text

```
column number <colnum>: if varaddr is NULL and varlen greater than 0, the table
column type must be SYBTEXT or SYBIMAGE and the program variable type must be
SYBTEXT, SYBCHAR, SYBIMAGE or SYBBINARY.
```

Possible Cause

`bcp_bind()` called incorrectly.

Action/solution

Correct application coding.

Additional information

Versions

All

6.226 20233

bcp_colfmt(): If table_colnum is 0, host_type cannot be 0.

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

6.227 20234

varlen should be greater than or equal to -1.

Symbolic constant

SYBEBCVLEN

Message text

```
varlen should be greater than or equal to -1.
```

Possible Cause

`bcp_bind()` or `bcp_collen()` has been called with a `<varlen>` value of less than -1.

Action/solution

Correct application coding.

Additional information

Versions

All

6.228 20235

host_collen should be greater than or equal to -1.

Symbolic constant

SYBEBCHLEN

Message text

```
host_collen should be greater than or equal to -1.
```

Possible Cause

Invalid value for `<host_colle><n>` passed to `bcp_colfmt_ps()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.229 20236

Illegal prefix length. Legal values are 0, 1, 2 or 4.

Symbolic constant

SYBEBCBPREF

Message text

```
Illegal prefix length. Legal values are 0, 1, 2 or 4.
```

Possible Cause

Invalid value for `<prefixle>`n passed to `bcp_bind()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.230 20237

Illegal prefix length. Legal values are 1, 0, 1, 2 or 4.

Symbolic constant

SYBEBCBPREF

Message text

```
Illegal prefix length. Legal values are 1, 0, 1, 2 or 4.
```

Possible Cause

Invalid value for `<host_prefixle>`n passed to `bcp_colfmt_ps()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.231 20238

bcp_init(): tblname parameter cannot be NULL.

Symbolic constant

SYBEBCITBNM

Message text

```
bcp_init(): tblname parameter cannot be NULL.
```

Possible Cause

`bcp_init()` called incorrectly.

Action/solution

Correct application coding.

Additional information

Versions

All

6.232 20239

bcp_init(): tblname parameter is too long.

Symbolic constant

SYBEBCITBLEN

Message text

```
bcp_init(): tblname parameter is too long.
```

Possible Cause

`bcp_init()` called incorrectly.

Action/solution

Correct application coding.

Additional information

Versions

All

6.233 20240

bcp_sendrow() may NOT be called unless all text data for the previous row has been sent using bcp_moretext().

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

6.234 20241

bcp protocol error: returned column count differs from the actual number of columns received.

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 SAP Technical Support.

Additional information

Versions

All

6.235 20242

bcp protocol error: expected default information and got none.

Symbolic constant

SYBEBPRODEF

Message text

```
bcp protocol error: expected default information and got none.
```

Possible Cause

Internal column mismatch.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.236 20243

bcp protocol error: expected number of defaults differs from the actual number of defaults received.

Symbolic constant

SYBEBPRONUMDEF

Message text

```
bcp protocol error: expected number of defaults differs from the actual number
of defaults received.
```

Possible Cause

Internal column default mismatch.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.237 20244

bcp protocol error: default column id and actual column id are not same.

Symbolic constant

SYBEBPRODEFID

Message text

```
bcp protocol error: default column id and actual column id are not same.
```

Possible Cause

Internal column default mismatch.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.238 20245

bcp protocol error: default value received for column that does not have default.

Symbolic constant

SYBEBPRONODEF

Message text

```
bcp protocol error: default value received for column that does not have default.
```

Possible Cause

Internal column default mismatch.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.239 20246

bcp protocol error: default value datatype differs from column datatype.

Symbolic constant

SYBEBPRODEFTYP

Message text

```
bcp protocol error: default value datatype differs from column datatype.
```

Possible Cause

Internal column default mismatch.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.240 20247

bcp protocol error: more than one row of default information received.

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 SAP Technical Support.

Additional information

Versions

All

6.241 20248

bcp protocol error: unexpected set of results received.

Symbolic constant

SYBEBPROEXTRES

Message text

```
bcp protocol error: unexpected set of results received.
```

Possible Cause

Extra results sent from server.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.242 20249

bcp protocol error: illegal default column id received.

Symbolic constant

SYBEBPROBADDEF

Message text

```
bcp protocol error: illegal default column id received.
```

Possible Cause

Internal column default mismatch.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.243 20250

bcp protocol error: unknown column datatype.

Symbolic constant

SYBEBPROBADTYP

Message text

```
bcp protocol error: unknown column datatype.
```

Possible Cause

Unknown datetype from server.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.244 20251

bcp protocol error: illegal datatype length received.

Symbolic constant

SYBEBPROBADLEN

Message text

```
bcp protocol error: illegal datatype length received.
```

Possible Cause

Illegal length received from the server.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.245 20252

bcp protocol error: illegal precision value received.

Symbolic constant

SYBEBPROBADPREC

Message text

```
bcp protocol error: illegal precision value received.
```

Possible Cause

Illegal precision received from server.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.246 20253

bcp protocol error: illegal scale value received.

Symbolic constant

SYBEBPROBADSCL

Message text

```
bcp protocol error: illegal scale value received.
```

Possible Cause

Illegal scale received from the server.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.247 20254

Illegal value for type parameter given to <routine>.

Symbolic constant

SYBEBADTYPE

Message text

```
Illegal value for type parameter given to <routine>.
```

Possible Cause

Invalid `<type>` passed to `dbsechandle()`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.248 20255

Cursor statement generated no results.

Symbolic constant

SYBECRSNORES

Message text

```
Cursor statement generated no results.
```

Possible Cause

Cursor statement returned no results.

Action/solution

No action necessary.

Additional information

Versions

All

6.249 20256

One of the tables involved in the cursor statement does not have a unique index

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

6.250 20257

A view cannot be joined with another table or a view in a cursor statement.

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

6.251 20258

The view used in the cursor statement does not include all the unique index columns of the underlying tables.

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

6.252 20259

Only fully keyset driven cursors can have 'order by', 'group by', or 'having' phrases.

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

6.253 20260

Row buffering should not be turned on when using cursor APIs.

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

6.254 20261

The DBNOAUTOFREE option should not be turned on when using cursor APIs.

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

6.255 20262

Cursor statement contains one of the disallowed phrases 'compute', 'union', 'for browse', or 'select into'.

Symbolic constant

SYBECRSDIS

Message text

```
Cursor statement contains one of the disallowed phrases 'compute', 'union', 'for
browse', or 'select into'.
```

Possible Cause

Invalid cursor statement.

Action/solution

Correct application coding.

Additional information

Versions

All

6.256 20263

Aggregate functions are not allowed in a cursor statement.

Symbolic constant

SYBECRSAGR

Message text

```
Aggregate functions are not allowed in a cursor statement.
```

Possible Cause

Invalid cursor statement.

Action/solution

Correct application coding.

Additional information

Versions

All

6.257 20264

Fetch types RANDOM and RELATIVE can only be used within the keyset of keyset driven cursors.

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

6.258 20265

Fetch type LAST requires fully keyset driven cursors.

Symbolic constant

SYBECRSFLAST

Message text

```
Fetch type LAST requires fully keyset driven cursors.
```

Possible Cause

`dbcursoropen()` called with `<scrollopt>` other than CUR_KEYSET.

Action/solution

Correct application coding.

Additional information

Versions

All

6.259 20266

Backward scrolling cannot be used in a forward scrolling cursor.

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

6.260 20267

Row number to be fetched is outside the valid range.

Symbolic constant

SYBECRSFROWN

Message text

```
Row number to be fetched is outside the valid range.
```

Possible Cause

Attempt to fetch a row before `<firstrow>` or after `<lastrow>`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.261 20268

Keyset cannot be scrolled backward in mixed cursors with a previous fetch type.

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 `<firstrow>` in a keyset-driven cursor.

Action/solution

Correct application coding.

Additional information

Versions

All

6.262 20269

Data locking or modifications cannot be made in a READONLY cursor.

Symbolic constant

SYBECRSRO

Message text

```
Data locking or modifications cannot be made in a READONLY cursor.
```

Possible Cause

`dbcursor()` called with `<optype>` other than CRS_REFRESH on a read-only cursor.

Action/solution

Correct application coding.

Additional information

Versions

All

6.263 20270

The DBNOCOUNT option should not be turned on when doing updates or deletes with dbcursor().

Symbolic constant

SYBECRSNOCOUNT

Message text

```
The DBNOCOUNT option should not be turned on when doing updates or deletes with
dbcursor().
```

Possible Cause

`DBNOCOUNT` option has been previously set with `dbsetopt()`. This is incompatible with cursor update/delete operations.

Action/solution

Correct application coding.

Additional information

Versions

All

6.264 20271

Table name must be determined in operations involving data locking or modifications

Symbolic constant

SYBECRSTAB

Message text

```
Table name must be determined in operations involving data locking or
modifications.
```

Possible Cause

`dbcursor()` is called with an invalid `<table>` value.

Action/solution

Correct application coding.

Additional information

Versions

All

6.265 20272

Update or insert operations cannot use bind variables when binding type is NOBIND.

Symbolic constant

SYBECRSUPDNB

Message text

```
Update or insert operations cannot use bind variables when binding type is
NOBIND.
```

Possible Cause

`dbcursorfetch()` has previously been called with vartype `<NOBIND>`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.266 20273

A WHERE clause is not allowed in a cursor update or insert.

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

6.267 20274

A SET clause is required for a cursor update or insert.

Symbolic constant

SYBECRSSET

Message text

```
A SET clause is required for a cursor update or insert.
```

Possible Cause

`dbcursor()` argument `<values>` must contain a SET clause.

Action/solution

Correct application coding.

Additional information

Versions

All

6.268 20275

Update or insert operations using bind variables require single table cursors.

Symbolic constant

SYBECRSUPDTAB

Message text

```
Update or insert operations using bind variables require single table cursors.
```

Possible Cause

`dbcursoropenI()` `stmt` argument affects multiple tables.

Action/solution

Correct application coding.

Additional information

Versions

All

6.269 20276

Update or delete operation did not affect any rows.

Symbolic constant

SYBECRSNOUPD

Message text

```
Update or delete operation did not affect any rows.
```

Possible Cause

`dbcursor()` with `<CRS_UPDATE>` or `<CRS_DELETE>` `<optype>` argument affected 0 rows.

Action/solution

Correct application coding.

Additional information

Versions

All

6.270 20277

Invalid cursor statement.

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

6.271 20278

The entire keyset must be defined for KEYSET type cursors.

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

6.272 20279

Cursor bind must be called prior to dbcursor invocation.

Symbolic constant

SYBECRSNOBIND

Message text

```
Cursor bind must be called prior to dbcursor invocation.
```

Possible Cause

`dbcursor()` called with NULL `<values>` argument.

Action/solution

Correct application coding

Additional information

Versions

All

6.273 20280

Unknown fetch type.

Symbolic constant

SYBECRSFTYPE

Message text

```
Unknown fetch type.
```

Possible Cause

`dbcursorfetch()` has been called with an unknown value for `<fetchtype>`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.274 20282

Multiple rows are returned, only one is expected while retrieving dbname.

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 SAP Technical Support.

Additional information

Versions

All

6.275 20283

No rows returned, at least one is expected.

Symbolic constant

SYBECRSNROWS

Message text

```
No rows returned, at least one is expected.
```

Possible Cause

Action/solution

Contact SAP Technical Support.

Additional information

Unexpected internal error.

Versions

All

6.276 20284

No unique index found.

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

6.277 20285

No OPTCC was found.

Symbolic constant

SYBECRSNOPTCC

Message text

```
No OPTCC was found.
```

Possible Cause

Could not find columns with CUR_OPTCC set.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.278 20286

The order of clauses must be ‘from’, ‘where’, and ‘order by’.

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

6.279 20287

Table name is NULL.

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

6.280 20288

No unique keys associated with this view.

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

6.281 20289

There is no valid address associated with this bind.

Symbolic constant

SYBECRSVAR

Message text

```
There is no valid address associated with this bind.
```

Possible Cause

`dbcursorbind()` called with `<vartype>` `<NOBIND>` and invalid `<pvaraddr>`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.282 20290

Security labels require both a name and a value.

Symbolic constant

SYBENOVALUE

Message text

```
Security labels require both a name and a value.
```

Possible Cause

Security label handler has set either a `<namelen>` or `<valuelen>` to a value <= 0.

Action/solution

Correct application coding.

Additional information

Versions

All

6.283 20291

Return parameter cannot be of the type SYBVOID.

Symbolic constant

SYBEVOIDRET

Message text

```
Return parameter cannot be of the type SYBVOID.
```

Possible Cause

`dbrpcparam()` attempted to define a return parameter as `<SYBVOI><D>`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.284 20292

Unable to close interfaces file.

Symbolic constant

SYBECLOSEIN

Message text

```
Unable to close interfaces file.
```

Possible Cause

Internal close failure.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.285 20293

Value of a boolean parameter should be either TRUE or FALSE.

Symbolic constant

SYBEBOOL

Message text

```
Value of a boolean parameter should be either TRUE or FALSE.
```

Possible Cause

`bcp_options()` has been called with option `<BCPLABELED>` and `<*value>` is not set to either TRUE or FALSE.

Action/solution

Correct application coding.

Additional information

Versions

All

6.286 20294

The <option> option cannot be called while a bulk copy operation is in progress

Symbolic constant

SYBEBCPOPT

Message text

```
The <option> option cannot be called while a bulk copy operation is in progress.
```

Possible Cause

`bcp_options()` with option `<BCPLABELED>` has been called when a bulk-copy operation is progress.

Action/solution

Correct application coding.

Additional information

Versions

All

6.287 20295

An illegal value was returned from the security label handler.

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 `<DMORELABEL>`, `<DBENDLABLE>`, or `<DBERRLABEL>`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.288 20296

Timed out waiting for server to ackowledge attention.

Symbolic constant

SYBEATTNACK

Message text

```
Timed out waiting for server to ackowledge attention.
```

Possible Cause

The server has not responded to a `dbcancel()` request.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.289 20297

Batch failed in bulk-copy to the server.

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 SAP Technical Support.

Additional information

Versions

All

6.290 20298

A directory control layer (DCL) error occurred.

Symbolic constant

SYBEDCL

Message text

```
A directory control layer (DCL) error occurred.
```

Possible Cause

Error reading directory service.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

All

6.291 20299

A CS Context error occurred.

Symbolic constant

SYBECS

Message text

```
A CS Context error occurred.
```

Possible Cause

Obsolete

Action/solution

Contact SAP Technical Support.

Additional information

Versions

None.

6.292 20300

An invalid value was used for SYBOCS_DBVERSION.

Symbolic constant

SYBEVERENV

Message text

```
An invalid value was used for SYBOCS_DBVERSION.
```

Possible Cause

`dbsetversion()` has been called with an invalid `<version>`.

Action/solution

Correct application coding.

Additional information

Versions

All

6.293 20301

dbcursoropen(): The multiplication of scrollopt and nrows results in overflow.

Symbolic constant

SYBCOPNOV

Message text

```
dbcursoropen(): The multiplication of scrollopt and nrows results in overflow.
```

Possible Cause

Action/solution

Contact SAP Technical Support.

Additional information

Internal cursor error.

Versions

15.7 and later

6.294 20302

DB-LIBRARY internal error: The arithmetic operation results in integer overflow

Symbolic constant

SYBEINTOVFL

Message text

```
DB-LIBRARY internal error: The arithmetic operation results in integer overflow.
```

Possible Cause

A buffer is not large enough for character set translation.

Action/solution

Contact SAP Technical Support.

Additional information

Versions

15.7 and later

Important Disclaimers and Legal Information

www.sap.com/contactsap

THE BEST RUN SAP