CHAPTER 3

The Database Tools Interface

About this chapter

This chapter describes how to use the database tools library that is provided with Adaptive Server Anywhere to add database management features to C or C++ applications.

Contents

Introduction to the database tools interface

Sybase Adaptive Server Anywhere includes Sybase Central and a set of command-line utilities for managing databases. These database management utilities carry out tasks such as backing up databases, creating databases, translating transaction logs to SQL, and so on.

Supported platforms

All the database management utilities use a shared library called the database tools library. It is supplied for each of the Windows 3.x, Windows 95, and Windows NT operating systems. The name of this library is operating system-dependent:

Operating system	DBTOOLS library name
Windows 95 and NT	*dbtool6.dll*
Windows 3.x	*dbtool6w.dll*

You can develop your own database management utilities, or incorporate database management features into your applications, by calling the database tools library. This chapter describes the interface to the database tools library. In this chapter, we assume you are familiar with how to call DLLs from the development environment you are using.

The database tools library has functions, or entry points, for each of the database management utilities. In addition, functions must be called before use of other database tools functions and when you have finished using other database tools functions.

The DBTOOLS.H header file

The DBTOOLS header file included with Adaptive Server Anywhere lists the entry points to the DBTOOLS library and also the structures used to pass information to and from the library. The *dbtools.h* file is installed into the *h* subdirectory under your installation directory. You should consult the *dbtools.h* file for the latest information about the entry points and structure members.

The DBTOOLS.H header file includes two other files:

- **sqlca.h** This is included for resolution of various macros, not for the SQLCA itself.
- **dllapi.h** Defines preprocessor macros for operating-system dependent and language-dependent macros.

Also, the *sqldef.h* header file includes error return values.

Using the database tools interface

This section provides an overview of how to develop applications that use the DBTools interface for managing databases.

Using the import libraries

In order to use the DBTools functions, you must link your application against a DBTools **import library** which contains the required function definitions.

Supported platforms

Import libraries are operating system-specific and can be compiler-specific. Import libraries for the DBTools interface are provided with Adaptive Server Anywhere, and can be found in the *lib* subdirectory of each operating system directory, under your installation directory. The provided DBTools import libraries are as follows:

Operating system	Compiler	Library
Windows NT/95	Watcom	*win32\dbtlstw.lib*
Windows NT/95	Microsoft	*win32\dbtlstM.lib*
Windows NT/95	Borland	*win32\dbtlstB.lib*
Windows 3.x	All	*win\dbtoolsw.lib*

Starting and finishing the DBTools library

Before using any other DBTools functions, you must call DBToolsInit. When you are finished using the DBTools DLL, you must call DBToolsFini.

The primary purpose of the DBToolsInit and DBToolsFini functions is to allow the DBTools dll to load the Adaptive Server Anywhere language DLL. The language DLL contains localized versions of all error messages and prompts that DBTools uses internally. If DBToolsFini is not called, the reference count of the language DLL is not decremented, and it will not be unloaded, so be careful to ensure there is a matched pair of DBToolsInit/DBToolsFini calls.

The following code fragment illustrates how to initialize and clean up DBTools:

```
// Declarations
a_dbtools_info  info;
short           ret;

//Initialize the a_dbtools_info structure
```

```
memset( &info, 0, sizeof( a_dbtools_info) );
info.errorrtn = (MSG_CALLBACK)MyErrorRtn;

// initialize DBTools
ret = DBToolsInit( &info );
if( ret != EXIT_OKAY ) {
   // DLL initialization failed
   …
}
// call some DBTools routines . . .
…
// cleanup the DBTools dll
DBToolsFini( &info );
```

Calling the DBTools functions

All the tools are run by first filling out a structure, and then calling a function (or **entry point**) in the DBTools DLL. Each entry point takes a pointer to a single structure as argument.

The following example shows how to use the DBBackup function. The example is for either Windows 95 or Windows NT operating systems.

```
// Initialize the structure
a_backup_db backup_info;
memset( &backup_info, 0, sizeof( backup_info ) );

// Fill out the structure
backup_info.version = DB_TOOLS_VERSION_NUMBER;
backup_info.output_dir = "C:\BACKUP";
backup_info.connectparms
="uid=dba;pwd=sql;dbf=asademo.db";
backup_info.startline = "DBENG6.EXE";
backup_info.confirmrtn = (MSG_CALLBACK) ConfirmRtn ;
backup_info.errorrtn = (MSG_CALLBACK) ErrorRtn ;
backup_info.msgrtn = (MSG_CALLBACK) MessageRtn ;
backup_info.statusrtn = (MSG_CALLBACK) StatusRtn ;
backup_info.backup_database = TRUE;

// start the backup
DBBackup( &backup_info );
```

For information about the members of the DBTools structures, see "DBTools structures" on page 99.

Using callback functions

Several elements in DBTools structures are of type MSG_CALLBACK. These are pointers to callback functions.

Uses of callback functions

Callback functions allow DBTools functions to return control of operation to the user's calling application. The DBTools library uses callback functions to handle messages sent to the user by the DBTools functions, for four purposes:

- **Confirmation** Called when an action needs to be confirmed by the user. For example, if the backup directory does not exist, the tools DLL asks if it needs to be created.
- **Error message** Called to handle a message when an error occurs, such as when an operation is out of disk space.
- **Information message** Called for the tools to display some message to the user (such as the name of the current table being backed up).
- **Status information** Called for the tools to display the status of an operation (such as the percentage done when unloading a table).

Assigning a callback function to a structure

In operating systems other than Windows 3.x, you can directly assign a callback routine to the structure. The following statement is an example using a backup structure:

```
backup_info.errorrtn = (MSG_CALLBACK) MyFunction
```

MSG_CALLBACK is defined in the *dllapi.h* header file supplied with Adaptive Server Anywhere. Tools routines can call back to the Calling application with messages that should be displayed in the appropriate user interface, whether that be a windowing environment, standard output on a character-based systems, or other user interface.

If you are developing for Windows 3.x, you must use the MakeProcInstance Windows API **thunking** function when assigning values to the elements. A thunking function changes 16-bit values to equivalent 32-bit values. The following statement is an example using a backup structure:

```
backup_info.errorrtn =
    (MSG_CALLBACK)MakeProcInstance(
    (FARPROC)MyFunction, hInst );
```

After you have finished with the DLL, you must free Callback thunk using the FreeProcInstance Windows API function:

```
FreeProcInstance( (FARPROC) backup_info.errorrtn );
```

If no function is assigned to the structure member, the message is ignored.

Confirmation callback function example

The following example confirmation routine asks the user to answer YES or NO to a prompt, and returns the user's selection:

```
extern short _callback ConfirmRtn(
      char far * question )
{
   int ret;
   if( question != NULL ) {
      ret = MessageBox( HwndParent, question,
      "Confirm", MB_ICONEXCLAMTION|MB_YESNO );
   }
   return( 0 );
}
```

Error callback function example

The following is an example of an error message handling routine, which displays the error message in a message box.

```
extern short _callback ErrorRtn(
      char far * errorstr )
{
   if( errorstr != NULL ) {
      ret = MessageBox( HwndParent, errorstr,
      "Backup Error", MB_ICONSTOP|MB_OK );
   }
   return( 0 );
}
```

Message callback function example

A common implementation of a message callback function outputs the message to the screen:

```
extern short _callback MessageRtn(
      char far * errorstr )
{
   if( messagestr != NULL ) {
   OutputMessageToWindow( messagestr );
   }
   return( 0 );
}
```

Status callback function example

A status callback routine is called when the tools needs to display the status of an operation (like the percentage done unloading a table). Again, a common implementation would just output the message to the screen:

```
extern short _callback StatusRtn(
      char far * statusstr )
{
   if( statusstr == NULL ) {
      return FALSE;
   }
   OutputMessageToWindow( statustr );
   return TRUE;
}
```

Version numbers and compatibility

Each structure has a member that indicates the version number. You should use this version member to hold the version of the DBTools library that your application was developed against. The current version of the DBTools library is included as the constant in the *dbtools.h* header file.

❖ **To assign the current version number to a structure:**

- Assign the version constant to the version member of the structure before calling the DBTools function. The following line assigns the current version to a backup structure:

```
backup_info.version = DB_TOOLS_VERSION_NUMBER;
```

Compatibility

The version number allows your application to continue working against newer versions of the DBTools library. The DBTools functions use the version number supplied by your application to allow the application to work, even if new members have been added to the DBTools structure.

Applications will not work against older versions of the DBTools library.

Using bit fields

Many of the DBTools structures use bit fields to hold Boolean information in a compact manner. For example, the backup structure has the following bit fields:

```
char            backup_database : 1;
char            backup_logfile  : 1;
char            backup_writefile: 1;
char            no_confirm   : 1;
char            quiet     : 1;
char            rename_log   : 1;
char            truncate_log : 1;
char            rename_local_log: 1;
```

Each bit field is one bit long, indicated by the 1 to the right of the colon in the structure declaration..

You assign an integer value of 0 or 1 to a bit field to pass Boolean information to the structure.

A DBTools example

The following program illustrates the use of the DBTools library in the context of a backup program:

```
# define WINNT

#include <stdio.h>
#include "windows.h"
#include "string.h"
#include "dbtools.h"

extern short _callback ConfirmCallBack(char far * str){
   if( MessageBox( NULL, str, "Backup",
       MB_YESNO|MB_ICONQUESTION ) == IDYES ) {
      return 1;
   }
   return 0;
}

extern short _callback MessageCallBack( char far * str){
   if( str != NULL ) {
      fprintf( stdout, "%s", str );
      fprintf( stdout, "\n" );
      fflush( stdout );
   }
   return 0;
}

extern short _callback StatusCallBack( char far * str ){
   if( str != NULL ) {
      fprintf( stdout, "%s", str );
      fprintf( stdout, "\n" );
      fflush( stdout );
   }
   return 0;
}

extern short _callback ErrorCallBack( char far * str ){
   if( str != NULL ) {
      fprintf( stdout, "%s", str );
      fprintf( stdout, "\n" );
      fflush( stdout );
   }
   return 0;
}
```

```
// Main entry point into the program.
int main( int argc, char * argv[] ){
   a_backup_db     backup_info;
   a_dbtools_info  dbtinfo;
   char         dir_name[ _MAX_PATH + 1];
   char         connect[ 256 ];
   HINSTANCE    hinst;
   FARPROC         dbbackup;
   FARPROC         dbtoolsinit;
   FARPROC         dbtoolsfini;

   // Always initialize to 0 so new versions
   //of the structure will be compatible.
   memset( &backup_info, 0, sizeof( a_backup_db ) );
   backup_info.version = DB_TOOLS_VERSION_6_0_01;
   backup_info.quiet = 0;
   backup_info.no_confirm = 0;
   backup_info.confirmrtn =
                        (MSG_CALLBACK)ConfirmCallBack;
   backup_info.errorrtn = (MSG_CALLBACK)ErrorCallBack;
   backup_info.msgrtn = (MSG_CALLBACK)MessageCallBack;
   backup_info.statusrtn = (MSG_CALLBACK)StatusCallBack;

   if( argc > 1 ) {
      strncpy( dir_name, argv[1], _MAX_PATH );
   } else {
      // DBTools does not expect (or like) the
      // trailing slash
      strcpy( dir_name, "c:\\temp" );
   }
   backup_info.output_dir = dir_name;

   if( argc > 2 ) {
      strncpy( connect, argv[2], 255 );
   } else {
      // Assume that the engine is already running.
      strcpy( connect, "UID=dba;PWD=sql;DBN=asademo" );
   }
   backup_info.connectparms = connect;
   backup_info.startline = "";
   backup_info.quiet = 0;
   backup_info.no_confirm = 0;
   backup_info.backup_database = 1;
   backup_info.backup_logfile = 1;
   backup_info.backup_writefile = 1;
   backup_info.rename_log = 0;
   backup_info.truncate_log = 0;
```

```
    hinst = LoadLibrary( "dbtool6.dll" );
    if( hinst == NULL ) {
       // Failed
       return 0;
    }
    dbtinfo.errorrtn = (MSG_CALLBACK)ErrorCallBack;
    dbbackup = GetProcAddress( (HMODULE)hinst,
       "_DBBackup@4" );
    dbtoolsinit = GetProcAddress( (HMODULE)hinst,
       "_DBToolsInit@4" );
    dbtoolsfini = GetProcAddress( (HMODULE)hinst,
       "_DBToolsFini@4" );
    (*dbtoolsinit)( &dbtinfo );
    (*dbbackup)( &backup_info );
    (*dbtoolsfini)( &dbtinfo );
    FreeLibrary( hinst );
    return 0;
}
```

DBTools functions

This section describes the functions available in the DBTools library. The functions are listed alphabetically.

DBBackup function

Function Database backup function. This function is used by the DBBACKUP command-line utility.

Prototype **short DBBackup** (**const** *backup_db* *);

Parameters

Parameter	Description
backup_db	Pointer to "a_backup_db structure" on page 99

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage The DBBackup function manages all database backup tasks. For descriptions of these tasks, see "The Backup utility" on page 67 of the book *Adaptive Server Anywhere Reference Manual*.

See Also "a_backup_db structure" on page 99

DBChangeLogName function

Function Changes the name of the transaction log file. This function is used by the *dblog* command-line utility.

Prototype **short DBChangeLogName** (**const** *change_log* *);

Parameters

Parameter	Description
change_log	Pointer to "a_change_log structure" on page 101

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage The -t option of the *dblog* command line utility changes the name of the transaction log. DBChangeLogName provides a programmatic interface to this function.

For descriptions of the *dblog* utility, see "The Transaction Log utility" on page 105 of the book *Adaptive Server Anywhere Reference Manual*.

See Also "a_change_log structure" on page 101

DBChangeWriteFile function

Function Changes a write file to refer to another database file. This function is used by the *dbwrite* command-line utility when the `-d` option is applied.

Prototype **short DBChangeWriteFile** (**const** *writefile* *);

Parameters

Parameter	Description
writefile	Pointer to "a_writefile structure" on page 120

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage For information about the Write File utility and its features, see "The Write File utility" on page 121 of the book *Adaptive Server Anywhere Reference Manual*.

See Also
"DBCreateWriteFile function" on page 91
"DBStatusWriteFile function" on page 94
"a_writefile structure" on page 120

DBCollate function

Function Extracts a collation sequence from a database.

Prototype **short DBCollate** (**const** *db_collation* *);

Parameters

Parameter	Description
db_collation	Pointer to "a_db_collation structure" on page 106

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage For information about the collation utility and its features, see "The Collation utility" on page 71 of the book *Adaptive Server Anywhere Reference Manual*

See Also "a_db_collation structure" on page 106

DBCompress function

Function Compresses a database file. This function is used by the *dbshrink* command-line utility.

Prototype **short DBCompress** (**const** *compress_db* *);

Parameters

Parameter	Description
compress_db	Pointer to "a_compress_db structure" on page 102

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage For information about the Compression utility and its features, see "The Compression utility" on page 75 of the book *Adaptive Server Anywhere Reference Manual*.

See Also "a_compress_db structure" on page 102

DBCreate function

Function Creates a database. This function is used by the *dbinit* command-line utility.

Prototype **short DBCreate** (**const** *create_db* *);

Parameters

Parameter	Description
create_db	Pointer to "a_create_db structure" on page 104

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage For information about the initialization utility, see "The Initialization utility" on page 84 of the book *Adaptive Server Anywhere Reference Manual*.

See Also "a_create_db structure" on page 104

DBCreateWriteFile function

Function Creates a write file. This function is used by the *dbwrite* command-line utility when the `-c` option is applied.

Prototype **short DBCreateWriteFile** (**const** *writefile* *);

Parameters

Parameter	Description
writefile	Pointer to "a_writefile structure" on page 120

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage For information about the Write File utility and its features, see "The Write File utility" on page 121 of the book *Adaptive Server Anywhere Reference Manual*.

See Also "DBChangeWriteFile function" on page 90
"DBStatusWriteFile function" on page 94
"a_writefile structure" on page 120

DBErase function

Function Erases a database file and/or transaction log file. This function is used by the *dberase* command-line utility.

Prototype **short DBErase** (**const** *erase_db* *);

Parameters

Parameter	Description
erase_db	Pointer to "an_erase_db structure" on page 109

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage For information about the Erase utility and its features, see "The Erase utility" on page 80 of the book *Adaptive Server Anywhere Reference Manual*.

See Also "an_erase_db structure" on page 109

DBExpand function

Function Uncompresses a database file. This function is used by the *dbexpand* command-line utility.

Prototype **short DBExpand** (**const** *expand_db* *);

Parameters

Parameter	Description
&an_expand_db	Pointer to "an_expand_db structure" on page 110

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage For information about the Uncompression utility and its features, see "The Uncompression utility" on page 108 of the book *Adaptive Server Anywhere Reference Manual*.

See Also "an_expand_db structure" on page 110

DBInfo function

Function Returns information about a database file. This function is used by the *dbinfo* command-line utility.

Prototype **short DBInfo** (**const** *db_info* *);

Parameters

Parameter	Description
db_info	Pointer to "a_db_info structure" on page 107

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage For information about the Information utility and its features, see "The Information utility" on page 82 of the book *Adaptive Server Anywhere Reference Manual*.

See Also "DBInfoDump function" on page 93
"DBInfoFree function" on page 93
"a_db_info structure" on page 107

DBInfoDump function

Function Returns information about a database file. This function is used by the *dbinfo* command-line utility when the `-u` option is used.

Prototype **short DBInfoDump** (**const** *db_info* *);

Parameters

Parameter	Description
db_info	Pointer to "a_db_info structure" on page 107

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage For information about the Information utility and its features, see "The Information utility" on page 82 of the book *Adaptive Server Anywhere Reference Manual*.

See Also "DBInfo function" on page 93
"DBInfoFree function" on page 93
"a_db_info structure" on page 107

DBInfoFree function

Function Called to free resources after the DBInfoDump function is called.

Prototype

short DBInfoFree (**const** *db_info* *);

Parameters

Parameter	Description
db_info	Pointer to "a_db_info structure" on page 107

Return value

A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage

For information about the Information utility and its features, see "The Information utility" on page 82 of the book *Adaptive Server Anywhere Reference Manual*.

See Also

"DBInfo function" on page 93
"DBInfoDump function" on page 93
"a_db_info structure" on page 107

DBStatusWriteFile function

Function

Gets the status of a write file. This function is used by the *dbwrite* command-line utility when the `-s` option is applied.

Prototype

short DBStatusWriteFile (**const** *writefile* *);

Parameters

Parameter	Description
writefile	Pointer to "a_writefile structure" on page 120

Return value

A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage

For information about the Write File utility and its features, see "The Write File utility" on page 121 of the book *Adaptive Server Anywhere Reference Manual*.

See Also

"DBChangeWriteFile function" on page 90
"DBCreateWriteFile function" on page 91
"a_writefile structure" on page 120

DBToolsFini function

Function

Decrements the counter and frees resources when an application is finished with the DBTools library.

Prototype

short DBToolsFini (**const** *dbtools_info* *);

Parameters

Parameter	Description
dbtools_info	Pointer to "a_dbtools_info structure" on page 108

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage The DBToolsFini function must be called at the end of any application that uses the DBTools interface. Failure to do so can lead to lost memory resources.

See Also "DBToolsInit function" on page 95
"a_dbtools_info structure" on page 108

DBToolsInit function

Function Prepares the DBTools library for use.

Prototype **short DBToolsInit t(const** *dbtools_info* *****);

Parameters

Parameter	Description
dbtools_info	Pointer to "a_dbtools_info structure" on page 108

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage The primary purpose of the DBToolsInit function is to load the Adaptive Server Anywhere language DLL. The language DLL contains localized versions of error messages and prompts that DBTools uses internally.

The DBToolsInit function must be called at the start of any application that uses the DBTools interface, before any other DBTools functions.

Example

- The following code sample illustrates how to initialize and clean up DBTools:

```
a_dbtools_info  info;
short        ret;

memset( &info, 0, sizeof( a_dbtools_info) );
info.errorrtn = (MSG_CALLBACK)MakeProcInstance(
          (FARPROC)MyErrorRtn, hInst );

// initialize DBTools
ret = DBToolsInit( &info );
if( ret != EXIT_OKAY ) {
      // DLL initialization failed
      …
```

```
}
// call some DBTools routines . . .
…
// cleanup the DBTools dll
DBToolsFini( &info );
```

See Also "DBToolsFini function" on page 94
"a_dbtools_info structure" on page 108

DBToolsVersion function

Function Returns the version number of the DBTools library.

Prototype **short DBToolsVersion** (*void*);

Return value A short integer indicating the version number of the DBTools library.

Usage Use the DBToolsVersion function to check that the DBTools library is not older than one against which your application is developed. While applications can run against newer versions of DBTools, they cannot run against older versions.

See Also "Version numbers and compatibility" on page 85

DBTranslateLog function

Function Translates a transaction log file to SQL. This function is used by the *dbtran* command-line utility.

Prototype **short DBTranslateLog** (**const** *translate_log* *);

Parameters

Parameter	Description
translate_log	Pointer to "a_translate_log structure" on page 113

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage For information about the log translation utility, see "The Log Translation utility" on page 98 of the book *Adaptive Server Anywhere Reference Manual*.

See Also "a_translate_log structure" on page 113

DBTruncateLog function

Function Truncates a transaction log file. This function is used by the *dbbackup* command-line utility.

Prototype
short DBTruncateLog (**const** *truncate_log* *);

Parameters

Parameter	Description
truncate_log	Pointer to "a_truncate_log structure" on page 114

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage For information about the backup utility, see "The Backup utility" on page 67 of the book *Adaptive Server Anywhere Reference Manual*

See Also "a_truncate_log structure" on page 114

DBUnload function

Function Unloads a database. This function is used by the *dbunload* command-line utility and also by the *dbxtract* utility for SQL Remote.

Prototype
short DBUnload (**const** *unload_db* *);

Parameters

Parameter	Description
an_unload_db	Pointer to "an_unload_db structure" on page 115

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage For information about the Unload utility, see "The Unload utility" on page 110 of the book *Adaptive Server Anywhere Reference Manual*.

See Also "an_unload_db structure" on page 115

DBUpgrade function

Function Upgrades a database file. This function is used by the *dbupgrade* command-line utility.

Prototype
short DBUpgrade (**const** *upgrade_db* *);

Parameters

Parameter	Description
upgrade_db	Pointer to "an_upgrade_db structure" on page 118

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage For information about the upgrade utility, see "The Upgrade utility" on page 116 of the book *Adaptive Server Anywhere Reference Manual*.

See Also "an_upgrade_db structure" on page 118

DBValidate function

Function Validates all or part of a database. This function is used by the *dbvalid* command-line utility.

Prototype **short DBValidate** (**const** *validate_db * *);

Parameters

Parameter	Description
validate_db	Pointer to "a_validate_db structure" on page 119

Return value A return code, as listed in "Software component Return codes" on page 65 of the book *Adaptive Server Anywhere Reference Manual*.

Usage For information about the upgrade utility, see "The Validation utility" on page 119 of the book *Adaptive Server Anywhere Reference Manual*.

See Also "a_validate_db structure" on page 119

DBTools structures

This section lists the structures that are used to exchange information with the DBTools library. The structures are listed alphabetically.

Many of the structure elements correspond to command-line switches on the corresponding utility. For example, several structures have a member named quiet, which can take on values of 0 or 1. This member corresponds to the quiet operation (`-q`) command-line option used by many of the utilities.

a_backup_db structure

Function

Holds the information needed to carry out backup tasks using the DBTools library.

Syntax

```
typedef struct a_backup_db {
   unsigned short  version;
   const char *    output_dir;
   const char *    connectparms;
   const char *    startline;
   MSG_CALLBACK    confirmrtn;
   MSG_CALLBACK    errorrtn;
   MSG_CALLBACK    msgrtn;
   MSG_CALLBACK    statusrtn;
   char            backup_database : 1;
   char            backup_logfile  : 1;
   char            backup_writefile: 1;
   char            no_confirm   : 1;
   char            quiet     : 1;
   char            rename_log   : 1;
   char            truncate_log : 1;
   char            rename_local_log: 1;
   const char *    hotlog_filename;
} a_backup_db;
```

Parameters

Member	Description
version	DBTools version number.
output_dir	Path to the output directory. For example: `"c:\backup"`
connectparms	Parameters needed to connect to the database. They take the form of connection strings, such as the following: `"UID=DBA;PWD=SQL;DBF=c:\asa6\asademo.db"` For the full range of connection string options, see "Connection parameters" on page 46 of the book *Adaptive Server Anywhere User's Guide*
startline	Command line used to start the database engine. The following is an example start line: `"c:\asa6\win32\dbeng6.exe"` The default start line is used if this member is NULL.
confirmrtn	Callback routine for confirming an action
errorrtn	Callback routine for handling an error message
msgrtn	Callback routine for handling an information message
statusrtn	Callback routine for handling a status message
backup_database	Backup the database file (1) or not (0)
backup_logfile	Backup the transaction log file (1) or not (0)
backup_writefile	Backup the database write file (1) or not (0), if a write file is being used
no_confirm	Operate with (0) or without (1) confirmation
quiet	Operate without printing messages (1), or print messages (0)
rename_log	rename the transaction log
truncate_log	delete the transaction log
rename_local_log	rename the local backup of the transaction log
hotlog_filename	file name for the live backup file

See Also

"DBBackup function" on page 89

For more information on callback functions, see "Using callback functions" on page 83.

a_change_log structure

Function

Holds the information needed to carry out *dblog* tasks using the DBTools library.

Syntax

```
typedef struct a_change_log {
unsigned short     version;
   const char *    dbname;
   const char *    logname;
   MSG_CALLBACK    errorrtn;
   MSG_CALLBACK    msgrtn;
   char      query_only         : 1;
   char      quiet              : 1;
   char      mirrorname_present : 1;
   char      change_mirrorname  : 1;
   char      change_logname     : 1;
   char      ignore_ltm_trunc: 1;
   char      ignore_remote_trunc: 1;
   char      set_generation_number : 1;
   const char *    mirrorname;
   unsigned short  generation_number;
} a_change_log;
```

Parameters

Member	Description
version	DBTools version number
dbname	Database file name
logname	The name of the transaction log. If set to NULL, there is no log
errorrtn	Callback routine for handling an error message
msgrtn	Callback routine for handling an information message
query_only	If 1, just display the name of the transaction log. If 0, permit changing of the log name
quiet	Operate without printing messages (1), or print messages (0)
mirrorname_present	Set to 1. Indicates that the version of DBTools is recent enough to support the mirrorname field
change_mirrorname	If 1, permit changing of the log mirror name
change_logname	If 1, permit changing of the transaction log name

Member	Description
ignore_ltm_trunc	When using the Log Transfer Manager, performs the same function as the dbcc settrunc('ltm', 'gen_id', n) Replication Server function: For information on dbcc, see your Replication Server documentation
ignore_remote_trunc	For SQL Remote. Resets the offset kept for the purposes of the DELETE_OLD_LOGS option, allowing transaction logs to be deleted when they are no longer needed
set_generation_number	When using the Log Transfer Manager, used after a backup is restored to set the generation number
mirrorname	The new name of the transaction log mirror file
generation_number	The new generation number. Used together with set_generation_number

See Also

"DBChangeLogName function" on page 89

For more information on callback functions, see "Using callback functions" on page 83.

a_compress_db structure

Function

Holds the information needed to carry out database compression tasks using the DBTools library.

Syntax

```
typedef struct a_compress_db {
   unsigned short  version;
   const char *    dbname;
   const char *    compress_name;
   MSG_CALLBACK    errorrtn;
   MSG_CALLBACK    msgrtn;
   MSG_CALLBACK    statusrtn;
   char            display_free_pages : 1;
   char            quiet              : 1;
   char            record_unchanged   : 1;
   a_compress_stats * stats;
   MSG_CALLBACK    confirmrtn;
   char            noconfirm    : 1;
} a_compress_db;
```

Parameters

Member	Description
version	DBTools version number.
dbname	The file name of the database to compress.
compress_name	The file name of the compressed database
errorrtn	Callback routine for handling an error message.
msgrtn	Callback routine for handling an information message.
statusrtn	Callback routine for handling a status message.
display_free_pages	Show the free page information.
quiet	Operate without printing messages (1), or print messages (0).
record_unchanged	Set to 1. Indicates that the a_compress_stats structure is recent enough to have an **unchanged** member.
a_compress_stats	Pointer to a structure of type a_compress_stats. This is filled in if the member is not NULL and display_free_pages is not zero.
confirmrtn	Callback routine for confirming an action.
noconfirm	Operate with (0) or without (1) confirmation.

See Also

"DBCompress function" on page 90
"a_compress_stats structure" on page 103
For more information on callback functions, see "Using callback functions" on page 83.

a_compress_stats structure

Function

Holds information describing compressed database file statistics.

Syntax

```
typedef struct a_compress_stats {
   a_stats_line    tables;
   a_stats_line    indices;
   a_stats_line    other;
   a_stats_line    free;
   a_stats_line    total;
   long            free_pages;
   long            unchanged;
} a_compress_stats;
```

Parameters

Member	Description
tables	Holds compression information regarding tables.
indices	Holds compression information regarding indexes
other	Holds other compression information
free	Holds information regarding free space.
total	Holds overall compression information.
free_pages	Holds information regarding free pages.
unchanged	The number of pages that the compression algorithm was unable to shrink.

See Also

"DBCompress function" on page 90
"a_compress_db structure" on page 102

a_create_db structure

Function

Holds the information needed to create a database using the DBTools library.

Syntax

```
typedef struct a_create_db {
   unsigned short     version;
   const char *       dbname;
   const char *       logname;
   const char *       startline;
   short              page_size;
   const char *       default_collation;
   MSG_CALLBACK       errorrtn;
   MSG_CALLBACK       msgrtn;
   short              database_version;
   char               verbose;
   char               blank_pad       : 2;
   char               respect_case    : 1;
   char               encrypt         : 1;
   char               debug           : 1;
   char               dbo_avail    : 1;
   char               mirrorname_present : 1;
   char               avoid_view_collisions : 1;
   short              collation_id;
   const char *       dbo_username;
   const char *       mirrorname;
} a_create_db;
```

Parameters

Member	Description
version	DBTools version number.
dbname	Database file name
logname	New transaction log name
startline	The command line used to start the database engine. The following is an example start line: `"c:\asa6\win32\dbeng6.exe"` The default start line is used if this member is NULL.
page_size	The page size of the database.
default_collation	The collation for the database.
errorrtn	Callback routine for handling an error message.
msgrtn	Callback routine for handling an information message.
database_version	The version number of the database.
verbose	Run in verbose mode.
blank_pad	Treat blanks as significant in string comparisons, and hold index information to reflect this.
respect_case	Make string comparisons case sensitive, and hold index information to reflect this.
encrypt	Encrypt the database.
debug	Reserved.
dbo_avail	Set to 1. The dbo user is available in this database.
mirrorname_present	Set to 1. Indicates that the version of DBTools is recent enough to support the mirrorname field.
avoid_view_collisions	Omit the generation of Watcom SQL compatibility views SYS.SYSCOLUMNS and SYS.SYSINDEXES.
collation_id	Collation identifier.
dbo_username	User name for the "dbo" user.
mirrorname	Transaction log mirror name.

See Also

"DBCreate function" on page 91

For more information on callback functions, see "Using callback functions" on page 83.

a_db_collation structure

Function

Holds the information needed to extract a collation sequence from a database using the DBTools library.

Syntax

```
typedef struct a_db_collation {
   unsigned short  version;
   const char *    connectparms;
   const char *    startline;
   const char *    collation_label;
   const char *    filename;
   MSG_CALLBACK    confirmrtn;
   MSG_CALLBACK    errorrtn;
   MSG_CALLBACK    msgrtn;
   char      include_empty      : 1;
   char      hex_for_extended: 1;
   char      replace            : 1;
   char      quiet              : 1;
} a_db_collation;
```

Parameters

Member	Description
version	DBTools version number.
connectparms	The parameters needed to connect to the database. They take the form of connection strings, such as the following: `"UID=DBA;PWD=SQL;DBF=c:\asa6\asademo.db"` For the full range of connection string options, see "Connection parameters" on page 46 of the book *Adaptive Server Anywhere User's Guide*
startline	The command line used to start the database engine. The following is an example start line: `"c:\asa6\win32\dbeng6.exe"` The default start line is used if this member is NULL.
confirmrtn	Callback routine for confirming an action.
errorrtn	Callback routine for handling an error message.
msgrtn	Callback routine for handling an information message.
include_empty	Write empty mappings for gaps in the collations sequence.
hex_for_extended	Use two-digit hexadecimal numbers to represent high-value characters.
replace	Operate without confirming actions.
quiet	Operate without messages.

See Also

"DBCollate function" on page 90

For more information on callback functions, see "Using callback functions" on page 83.

a_db_info structure

Function

Holds the information needed to return *dbinfo* information using the DBTools library.

Syntax

```
typedef struct a_db_info {
   unsigned short     version;
   MSG_CALLBACK       errorrtn;
   const char *       dbname;
   unsigned short     dbbufsize;
   char *          dbnamebuffer;
   unsigned short     logbufsize;
   char *          lognamebuffer;
   unsigned short     wrtbufsize;
   char *          wrtnamebuffer;
   char               quiet : 1;
   char               mirrorname_present : 1;
   a_sysinfo          sysinfo;
   unsigned long      free_pages;
   unsigned char      compressed : 1;
   const char *       connectparms;
   const char *       startline;
   MSG_CALLBACK       msgrtn;
   MSG_CALLBACK       statusrtn;
   unsigned char      page_usage : 1;
   a_table_info *     totals;
   unsigned long      file_size;
   unsigned long      unused_pages;
   unsigned long      other_pages;
   unsigned short     mirrorbufsize;
   char *          mirrornamebuffer;
} a_db_info;
```

Parameters

Member	Description
version	DBTools version number.
errortrn	Callback routine for handling an error message.
dbname	Database file name.
dbbufsize	The length of the dbnamebuffer member.
dbnamebuffer	Database file name.
logbufsize	The length of the lognamebuffer member.
lognamebuffer	Transaction log file name.
wrtbufsize	The length of the wrtnamebuffer member.

Member	Description
wrtnamebuffer	The write file name.
quiet	Operate without confirming messages.
mirrorname_present	Set to 1. Indicates that the version of DBTools is recent enough to support the mirrorname field.
sysinfo	Pointer to a_sysinfo structure.
free_pages	Number of free pages.
compressed	1 if compressed, otherwise 0.
connectparms	The parameters needed to connect to the database. They take the form of connection strings, such as the following: `"UID=DBA;PWD=SQL;DBF=c:\sa50\asademo.db"` ☞ For the full range of connection string options, see "Connection parameters" on page 46 of the book *Adaptive Server Anywhere User's Guide*.
startline	The command line used to start the database engine. The following is an example start line: `"c:\asa6\win32\dbeng6.exe"` The default start line is used if this member is NULL.
msgrtn	Callback routine for handling an information message.
statusrtn	Callback routine for handling a status message.
page_usage	1 to report page usage statistics, otherwise 0.
totals	Pointer to a_table_info structure.
file_size	Size of database file.
unused_pages	Number of unused pages.
other_pages	Number of pages that are neither table nor index pages.
mirrorbufsize	The length of the mirrornamebuffer member.
mirrornamebuffer	The transaction log mirror name.

See Also

"DBInfo function" on page 93

☞ For more information on callback functions, see "Using callback functions" on page 83.

a_dbtools_info structure

Function

Holds the information needed to start and finish working with the DBTools library.

Syntax

```
typedef struct a_dbtools_info {
   MSG_CALLBACK    errorrtn;
} a_dbtools_info;
```

Parameters

Member	Description
errorrtn	Callback routine for handling an error message.

See Also

"DBToolsFini function" on page 94
"DBToolsInit function" on page 95

For more information on callback functions, see "Using callback functions" on page 83.

an_erase_db structure

Function

Holds information needed to erase a database using the DBTools library.

Syntax

```
typedef struct an_erase_db {
   unsigned short     version;
   const char *       dbname;
   MSG_CALLBACK       confirmrtn;
   MSG_CALLBACK       errorrtn;
   MSG_CALLBACK       msgrtn;
   char               quiet : 1;
   char               erase : 1;
} an_erase_db;
```

Parameters

Member	Description
version	DBTools version number.
dbname	Database file name to erase.
confirmrtn	Callback routine for confirming an action.
errorrtn	Callback routine for handling an error message.
msgrtn	Callback routine for handling an information message.
quiet	Operate without printing messages (1), or print messages (0).
erase	Erase without confirmation (1) or with confirmation (0).

See Also

"DBErase function" on page 92

For more information on callback functions, see "Using callback functions" on page 83.

an_expand_db structure

Function

Holds information needed for database expansion using the DBTools library.

Syntax

```
typedef struct an_expand_db {
   unsigned short     version;
   const char *       compress_name;
   const char *       dbname;
   MSG_CALLBACK       errorrtn;
   MSG_CALLBACK       msgrtn;
   MSG_CALLBACK       statusrtn;
   char               quiet : 1;
   MSG_CALLBACK       confirmrtn;
   char               noconfirm    : 1;
} an_expand_db;
```

Parameters

Member	Description
version	DBTools version number.
compress_name	Name of compressed database file
dbname	Database file name
errorrtn	Callback routine for handling an error message.
msgrtn	Callback routine for handling an information message.
statusrtn	Callback routine for handling a status message.
quiet	Operate without printing messages (1), or print messages (0).
confirmrtn	Callback routine for confirming an action.
noconfirm	Operate with (0) or without (1) confirmation.

See Also

"DBExpand function" on page 92

For more information on callback functions, see "Using callback functions" on page 83.

a_name structure

Function

Holds a linked list of names. This is used by other structures requiring lists of names.

Syntax

```
typedef struct a_name {
   struct a_name * next;
   char         name[1];
} a_name, * p_name;
```

Parameters

Member	Description
next	Pointer to the next a_name structure in the list
name	The name.
p_name	Pointer to the previous a_name structure

See Also

"a_translate_log structure" on page 113
"a_validate_db structure" on page 119
"an_unload_db structure" on page 115

a_stats_line structure

Function

Holds information needed for database compression and expansion using the DBTools library.

Syntax

```
typedef struct a_stats_line {
   long     pages;
   long     bytes;
   long     compressed_bytes;
} a_stats_line;
```

Parameters

Member	Description
pages	Number of pages.
bytes	Number of bytes for uncompressed database.
compressed_bytes	Number of bytes for compressed database.

See Also

"a_compress_stats structure" on page 103

a_sysinfo structure

Function

Holds information needed for *dbinfo* and *dbunload* utilities using the DBTools library.

Syntax

```
typedef struct a_sysinfo {
   char     valid_data   : 1;
   char     blank_padding   : 1;
   char     case_sensitivity: 1;
   char     encryption      : 1;
   char     default_collation[11];
   unsigned short  page_size;
} a_sysinfo;
```

Parameters

Member	Description
valid_date	Bit-field indicating whether the following values are set.
blank_padding	1 if blank padding is used in this database, 0 otherwise.
case_sensitivity	1 if the database is case-sensitive, 0 otherwise.
encryption	1 if the database is encrypted, 0 otherwise.
default_collation	The collation sequence for the database.
page_size	The page size for the database.

See Also "a_db_info structure" on page 107

a_table_info structure

Function Holds information about a table needed as part of the a_db_info structure.

Syntax

```
typedef struct a_table_info {
   struct a_table_info * next;
   unsigned short  table_id;
   unsigned long   table_pages;
   unsigned long   index_pages;
   unsigned long   table_used;
   unsigned long   index_used;
   char *     table_name;
} a_table_info;
```

Parameters

Member	Description
next	Next table in the list
table_id	ID number for this table
table_pages	Number of table pages
index_pages	Number of index pages
table_used	Number of bytes used in table pages
index_used	Number of bytes used in index pages
table_name	Name of the table

See Also "a_db_info structure" on page 107

a_translate_log structure

Function

Holds information needed for transaction log translation using the DBTools library.

Syntax

```
typedef struct a_translate_log {
   unsigned short  version;
   const char *    logname;
   const char *    sqlname;
   p_name       userlist;
   MSG_CALLBACK    confirmrtn;
   MSG_CALLBACK    errorrtn;
   MSG_CALLBACK    msgrtn;
   char            userlisttype;
   char            remove_rollback : 1;
   char            ansi_sql  : 1;
   char            since_checkpoint: 1;
   char            omit_comments   : 1;
   char            replace      : 1;
   char            debug     : 1;
   char            include_trigger_trans : 1;
   char            comment_trigger_trans : 1;
   unsigned long   since_time;
} a_translate_log;
```

Parameters

Member	Description
version	DBTools version number.
logname	The name of the transaction log file.
sqlname	The name of the SQL command file.
userlist	Pointer to a linked list of users.
confirmrtn	Callback routine for confirming an action.
errorrtn	Callback routine for handling an error message.
msgrtn	Callback routine for handling an information message.
userlisttype	A member of the "dbtran_userlist_type enumeration" on page 123.
remove_rollback	1 to remove changes that have been rolled back. 0 otherwise.
ansi_sql	If set to 0, generate UPDATE statements with a non-standard FIRST keyword. This is for cases where there is no primary key or unique index on a table, in case of duplicate rows. If set to 1, do not output this keyword.
since_checkpoint	If 1, translate entries since the last checkpoint only.

Member	Description
omit_comments	If 1, omit comments.
replace	Replace files without prompting for confirmation.
debug	Reserved.
include_trigger_trans	If 1, include trigger actions.
comment_trigger_trans	If 1, include trigger actions as comments in the SQL file.
since_time	Translate entries only since the specified time.

See Also

"DBTranslateLog function" on page 96
"a_name structure" on page 110
"dbtran_userlist_type enumeration" on page 123

For more information on callback functions, see "Using callback functions" on page 83.

a_truncate_log structure

Function

Holds information needed for transaction log truncation using the DBTools library.

Syntax

```
typedef struct a_truncate_log {
   unsigned short  version;
   const char *    connectparms;
   const char *    startline;
   MSG_CALLBACK    errorrtn;
   MSG_CALLBACK    msgrtn;
   char            quiet : 1;
} a_truncate_log;
```

Parameters

Member	Description
version	DBTools version number.
connectparms	The parameters needed to connect to the database. They take the form of connection strings, such as the following: `"UID=DBA;PWD=SQL;DBF=c:\asa6\asademo.db"` For the full range of connection string options, see "Connection parameters" on page 46 of the book *Adaptive Server Anywhere User's Guide*
startline	The command line used to start the database engine. The following is an example start line: `"c:\asa6\win32\dbeng6.exe"` The default start line is used if this member is NULL.
errorrtn	Callback routine for handling an error message.
msgrtn	Callback routine for handling an information message.
quiet	Operate without printing messages (1), or print messages (0).

See Also

"DBTruncateLog function" on page 96

For more information on callback functions, see "Using callback functions" on page 83.

an_unload_db structure

Function

Holds information needed to unload a database using the DBTools library or extract a remote database for SQL Remote. Those fields used by the *dbxtract* SQL Remote extraction utility are indicated.

Syntax

```
typedef struct an_unload_db {
   unsigned short     version;
   const char *       connectparms;
   const char *       startline;
   const char *       temp_dir;
   const char *       reload_filename;
   MSG_CALLBACK       errorrtn;
   MSG_CALLBACK       msgrtn;
   MSG_CALLBACK       statusrtn;
   MSG_CALLBACK       confirmrtn;
   char               unload_type;
   char               verbose;
   char               unordered : 1;
   char               no_confirm : 1;
   char               use_internal_unload : 1;
   char               dbo_avail : 1;
```

```
    char              extract : 1;
    char              table_list_provided : 1;
    char              exclude_tables : 1;
    char              more_flag_bits_present : 1;
    a_sysinfo         sysinfo;
    const char *      remote_dir;
    const char *      dbo_username;
    const char *      subscriber_username;
    const char *      publisher_address_type;
    const char *      publisher_address;
    unsigned short    isolation_level;
    char              start_subscriptions : 1;
    char              exclude_foreign_keys : 1;
    char              exclude_procedures : 1;
    char              exclude_triggers : 1;
    char              exclude_views : 1;
    char              isolation_set : 1;
    char              include_where_subscribe : 1;
    p_name          table_list;
    unsigned short    escape_char_present : 1;
    unsigned short    view_iterations_present : 1;
    unsigned short    view_iterations;
    char              escape_char;
} an_unload_db;
```

Parameters

Member	Description
version	DBTools version number.
connectparms	The parameters needed to connect to the database. They take the form of connection strings, such as the following: `"UID=DBA;PWD=SQL;DBF=c:\asa6\asademo.db"` For the full range of connection string options, see "Connection parameters" on page 46 of the book *Adaptive Server Anywhere User's Guide*.
startline	The command line used to start the database engine. The following is an example start line: `"c:\asa6\win32\dbeng6.exe"` The default start line is used if this member is NULL.
confirmrtn	Callback routine for confirming an action.
errorrtn	Callback routine for handling an error message.
msgrtn	Callback routine for handling an information message.
statusrtn	Callback routine for handling a status message.

Member	Description
unload_type	A member of the "dbunload type enumeration" on page 123
verbose	Operate in verbose mode.
unordered	If 1, output the data unordered.
no_confirm	If 1, operate without confirming actions.
use_internal_unload	If 1, use the UNLOAD TABLE unloading. Of 0, use the Interactive SQL OUTPUT statement.
dbo_avail	Set to 1. Indicates that the version of DBTools is recent enough to support the dbo_username field.
extract	If 1 carry out a SQL Remote extraction. Otherwise carry out an unload.
table_list_provided	If 1, a table list is provided. Unload data for those tables only.
exclude_tables	If 1, exclude data for the listed tables.
more_flag_bits_present	Set to 1. Indicates that the version of DBTools is recent enough to support the dbo_username field.
sysinfo	Pointer to a_sysinfo structure.
remote_dir	If extracting a database, the directory for remote users.
dbo_username	Username for the dbo user.
subscriber_username	If extracting a database, the user name of the subscriber.
publisher_address_type	If extracting a database, the message type for the publisher.
publisher_address	If extracting a database, the address for the publisher.
isolation_level	If extracting a database, perform the extraction at the specified isolation level.
start_subscriptions	If extracting a database, start the subscription.
exclude_foreign_keys	If extracting a database, exclude foreign key definitions.
exclude_procedures	If extracting a database, exclude procedures.
exclude_triggers	If extracting a database, exclude triggers.
exclude_views	If extracting a database, exclude views.
isolation_set	Set to 1 if the isolation level is set
include_where_subscribe	If extracting a database, extract fully qualified publications and subscriptions.

Member	Description
table_list	Pointer to a linked list of table names.
escape__present	Set to 1 if a non-default escape character is being specified.
view_iterations_present	Set to 1. Indicates that the version of DBTools is recent enough to support the view iterations option.
view_iterations	If your database contains view definitions that are dependent on each other, use this option to avoid failure in reloading the views into a database. The option causes view creation statements to be unloaded multiple times, as specified by the count entered.
escape_char	Escape character.

See Also

"DBUnload function" on page 97
"a_name structure" on page 110
"dbunload type enumeration" on page 123

☞ For more information on callback functions, see "Using callback functions" on page 83.

an_upgrade_db structure

Function

Holds information needed to upgrade a database using the DBTools library.

Syntax

```
typedef struct an_upgrade_db {
   unsigned short     version;
   const char *       connectparms;
   const char *       startline;
   MSG_CALLBACK       errorrtn;
   MSG_CALLBACK       msgrtn;
   MSG_CALLBACK       statusrtn;
   char               quiet : 1;
   char               dbo_avail : 1;
   const char *       dbo_username;
} an_upgrade_db;
```

Parameters

Member	Description
version	DBTools version number.
connectparms	The parameters needed to connect to the database. They take the form of connection strings, such as the following: `"UID=DBA;PWD=SQL;DBF=c:\asa6\asademo.db"` For the full range of connection string options, see "Connection parameters" on page 46 of the book *Adaptive Server Anywhere User's Guide*.
startline	The command line used to start the database engine. The following is an example start line: `"c:\asa6\win32\dbeng6.exe"` The default start line is used if this member is NULL.
errorrtn	Callback routine for handling an error message.
msgrtn	Callback routine for handling an information message.
statusrtn	Callback routine for handling a status message.
quiet	Operate without printing messages (1), or print messages (0).
dbo_avail	Set to 1. Indicates that the version of DBTools is recent enough to support the dbo_username field.
dbo_username	The name to use for the dbo.

See Also

"DBUpgrade function" on page 97

For more information on callback functions, see "Using callback functions" on page 83.

a_validate_db structure

Function

Holds information needed for database validation using the DBTools library.

Syntax

```
typedef struct a_validate_db {
   unsigned short     version;
   const char *       connectparms;
   const char *       startline;
   p_name          tables;
   MSG_CALLBACK       errorrtn;
   MSG_CALLBACK       msgrtn;
   MSG_CALLBACK       statusrtn;
   char               quiet : 1;
} a_validate_db;
```

Parameters

Member	Description
version	DBTools version number.
connectparms	The parameters needed to connect to the database. They take the form of connection strings, such as the following: `"UID=DBA;PWD=SQL;DBF=c:\asa6\asademo.db"` For the full range of connection string options, see "Connection parameters" on page 46 of the book *Adaptive Server Anywhere User's Guide*.
startline	The command line used to start the database engine. The following is an example start line: `"c:\asa6\win32\dbeng6.exe"` The default start line is used if this member is NULL.
tables	Pointer to a linked list of table names.
errorrtn	Callback routine for handling an error message.
msgrtn	Callback routine for handling an information message.
statusrtn	Callback routine for handling a status message.
quiet	Operate without printing messages (1), or print messages (0).

See Also

"DBValidate function" on page 98
"a_name structure" on page 110

For more information on callback functions, see "Using callback functions" on page 83.

a_writefile structure

Function

Holds information needed for database write file management using the DBTools library.

Syntax

```
typedef struct a_writefile {
unsigned short     version;
   const char *    writename;
   const char *    wlogname;
   const char *    dbname;
   const char *    forcename;
   MSG_CALLBACK    confirmrtn;
   MSG_CALLBACK    errorrtn;
   MSG_CALLBACK    msgrtn;
   char            action;
   char            quiet   : 1;
   char            erase   : 1;
   char            force   : 1;
```

```
    char            mirrorname_present : 1;
    const char *    wlogmirrorname;
} a_writefile;
```

Parameters

Member	Description
version	DBTools version number.
writename	Write file name.
wlogname	Used only when creating write files.
dbname	Used when changing and creating write files.
forcename	Forced file name reference.
confirmrtn	Callback routine for confirming an action. Only used when creating a write file.
errorrtn	Callback routine for handling an error message.
msgrtn	Callback routine for handling an information message.
action	Reserved for use by Sybase.
quiet	Operate without printing messages (1), or print messages (0).
erase	Used for creating write files only. Erase without confirmation (1) or with confirmation (0).
force	If 1, force the write file to point to a named file.
mirrorname_present	Used when creating only. Set to 1. Indicates that the version of DBTools is recent enough to support the mirrorname field.
wlogmirrorname	Name of the transaction log mirror.

See Also

"DBChangeWriteFile function" on page 90
"DBCreateWriteFile function" on page 91
"DBStatusWriteFile function" on page 94
For more information on callback functions, see "Using callback functions" on page 83.

DBTools enumeration types

This section lists the enumeration types that are used by the DBTools library. The enumerations are listed alphabetically.

Verbosity enumeration

Function

Specifies the volume of output.

Syntax

```
enum {
   VB_QUIET,
   VB_NORMAL,
   VB_VERBOSE
};
```

Parameters

Value	Description
VB_QUIET	No output
VB_NORMAL	Normal amount of output
VB_VERBOSE	Verbose output, useful for debugging.

See Also

"a_create_db structure" on page 104
"an_unload_db structure" on page 115

Blank padding enumeration

Function

Used in the "a_create_db structure" on page 104, to specify the value of blank_pad.

Syntax

```
enum {
   NO_BLANK_PADDING,
   BLANK_PADDING
};
```

Parameters

Value	Description
NO_BLANK_PADDING	Does not use blank padding
BLANK_PADDING	Uses blank padding

See Also

"a_create_db structure" on page 104

dbtran_userlist_type enumeration

Function The type of a user list, as used by an "a_translate_log structure" on page 113.

Syntax

```
typedef enum dbtran_userlist_type {
   DBTRAN_INCLUDE_ALL,
   DBTRAN_INCLUDE_SOME,
   DBTRAN_EXCLUDE_SOME
} dbtran_userlist_type;
```

Value	Description
DBTRAN_INCLUDE_ALL	Include operations from all users
DBTRAN_INCLUDE_SOME	Include operations only from the users listed in the supplied user list.
DBTRAN_EXCLUDE_SOME	Exclude operations from the users listed in the supplied user list.

See Also "a_translate_log structure" on page 113

dbunload type enumeration

Function The type of unload being performed, as used by the "an_unload_db structure" on page 115.

Syntax

```
enum {
   UNLOAD_ALL,
   UNLOAD_DATA_ONLY,
   UNLOAD_NO_DATA
};
```

Value	Description
UNLOAD_ALL	Unload both data and schema.
UNLOAD_DATA_ONLY	Unload data. Do not unload schema.
UNLOAD_NO_DATA	Unload schema only.

See Also "an_unload_db structure" on page 115