New Features
Adaptive Server® Enterprise 15.0.2 ESD#1

Document ID: DC00650-01-1502-01

Last revised: September 2007

This bulletin describes features in Adaptive Server that have been implemented since the publication of the *15.0.2 New Features Guide*.

Note Sybase Real-Time Data Service (RTDS) version 4.0 is not certified on Adaptive Server 15.0.2. If you have RTDS 4.0, and you upgrade Adaptive Server to version 15.0.2, RTDS messaging stops. For more information and workarounds, see the *Adaptive Server 15.0.2 Release Bulletin* for your platform.

sp_helptext enhancements

sp_helptext has been enhanced to regenerate formatted SQL text for compiled objects, identical to the SQL text for the compiled object when it was created. This also allows the user to generate a fragment of SQL source by providing a starting line number and a context block window that specifies the number of lines displayed. For complete documentation of sp_helptext, see the *Reference Manual: Procedures*. For information about the new parameters, see “sp_helptext” on page 8.

sp_monitor enhancements

In Adaptive Server® Enterprise 15.x, the stored procedure sp_monitor has been enhanced to support deadlock and procstack monitoring types. For syntax information about these parameters, see “System changes for sp_monitor” on page 10.

To summarize these enhancements:

- enable and disable commands are granular; they turn individual monitoring types on or off.
- A new command, list, displays the currently active monitoring type, the currently enabled configuration options, and any required configuration options that are not yet turned on.
- archive and report commands save monitoring data to an archive, and let you create reports from the archived data.
- The help command is enhanced to display detailed usage information, where possible with examples.

Enable and *disable*

The enable/disable options in sp_monitor have been enhanced to provide a greater level of granularity, including an argument that allows you to selectively enable or disable a specific monitoring type, and thus to set or unset the configuration options required by the specific monitoring type.

The default behavior is that all the monitoring types supported by sp_monitor are turned either on or off. Most configuration options required to support the monitoring types are also enabled or disabled.

If you do use enable or disable to specify an entity type, monitoring is enabled or disabled for only that particular entity, and the configuration options required for only that entity are turned ON or OFF.

deadlock

sp_monitor enhancements can fetch data from the monDeadLock table and present it to the user in a human-readable format. Sites that can neither turn on the configuration option print deadlock information, nor alter their applications to turn the deadlock traces on, can capture deadlock events directly from the monitoring table. Users can instead monitor deadlocks using sp_monitor, and use the results to tune and troubleshoot the areas of their applications that cause deadlocks.

You can use these features of this enhancement to:

- Create a default collection of all deadlock information from the monDeadLock table and printing out a block of output for each deadlock instance.

- Print out deadlock tracing only for a given deadlock ID.
- Print out deadlock tracing for deadlocks resolved on a specified day.
- Archive the deadlock event data from monDeadLock to a user-supplied archive table name:
  - Use select into to create the archive table, if it does not already exist
  - Insert new rows into the archive table.
  - Ensure that duplicate deadlock events are not reinserted to the archive table.
- Process deadlock event data from an archive, and generating deadlock tracing for all deadlock events, or for specified deadlock IDs, or for deadlocks resolved on a specified day.
- Provide output modes that summarize the frequency of deadlock events, whether grouped by date of event, application name, or table name.
- Support verbose output mode, to generate such details of the deadlock data as names of users, applications, or details about a procedure's identity.

Note These features are supported whether the deadlock analysis is performed directly on the monitoring table or on data archived in an archive table.

procstack

sp_monitor procstack examines the execution of a task within a nested stored procedure execution. The procedures executed are extracted by retrieving data from the monProcessProcedures table at runtime.

sp_monitor procstack lets you monitor connections by generating a SQL text fragment for each executing procedure, and a backtrace for a task that might, for example, be waiting for a lock while executing a deeply nested stored procedure sequence.

procstack generates a summary for the sequence of stored procedures executed, and calls sp_helptext to extract a SQL context block around the line number of each procedure's execution frame. If possible, the execution plan of the currently executed statement in the procedure is generated.

sp_monitor procstack can help you identify, during application development, errors that lead to blocking; during production, this interface regenerates inefficient SQL code at the moment performance slows down, or tasks are blocked.

list

list provides a snapshot of the monitoring enabled in the server, and of the configuration options required for the enabled monitoring entities. Any configuration options that are missing, but required by the enabled monitoring entities, are also listed. Configuration options that are already enabled, through either sp_configure or the configuration file, are also noted in the list output.

archive and *report*

archive and report commands display monitoring data for a specified monitoring type. These commands specify the location of the archive table.

For example:

```
sp_monitor archive, 'deadlock'

sp_monitor report, 'deadlock'
```

```
sp_monitor 'archive [using prefix=<string>]', {'<monitoring type>'}

sp_monitor 'report [using prefix=<string>]', '<monitoring type>'
      [,<options supported for monitoring_type>]
```

To report deadlock event data from a default archive table, monDeadLock, in the current database, mondb, enter:

```
sp_monitor report, 'deadlock'
```

```
sp_monitor 'archive [using prefix=<string>]', {'<monitoring type>'}
```

Usage :

- Periodically archive events.
- The monDeadLock pipe can overflow its size; Sybase recommends that you frequently archive deadlock events when they occur rapidly.
- If the pipe is not full, or has not rolled over, archiving stores only new event information.
- select into must be turned ON in the database where information is archived.

Note Currently, archiving and reporting are supported only for deadlock monitoring data from monDeadLock.

---

help

This command produces enhanced command-specific help and usage information for sp_monitor.

Enabling ascending inserts (*ascinserts*)

The ascinserts property of a table allows customers to insert records into a table in sorted order. If tables occasionally experience random inserts and have more ordered inserts during batch jobs, enable ascinserts only for the period the batch runs.

sp_chgattribute

Enable the ascinserts option using:

sp_chgattribute "*objname*", ascinserts, 0 |1

For example, to enable ascinserts for the titles table:

```
sp_chgattribute "titles", ascinserts, 1
```

sp_help

sp_help includes the value of ascinserts for any all-pages-locked table in its output.

This value is stored by bit 6 of the status2 column in sysindexes, and a value of 64 for this column indicates that ascinserts is enabled for this table.

Note ascinserts has also been added to dbcctune().

IPv6 platform support

IPv6 is now supported on IBM AIX.

New sort orders for Chinese character sets

This section describes two sort orders, gbpinyin and gbinyinnocs, supported in Adaptive Server version 15.0.2 ESD#1. These sort orders support these Chinese character sets:

- EUC-GB
- GB-18030
- CP-936
- UTF-8

GBPINYIN is a Chinese phonetic ordering, where the order is based on Chinese Pinyin pronunciation.

GBPINYINNOCS is the combination of GBPINYIN and NO-CASE ordering, in which Chinese characters are ordered by Pinyin pronunciation, and Latin characters are ordered as case-insensitive.

Use these stored procedures:

- sp_helpsort
- sp_configure

To populate the syscharsets table with the new sort orders:

```
select sortkey (null, 'all')
```

To list the new sort order ID:

```
sp_helpsort
------------
Name                      ID
-------------------------
gbpinyin                 163
gbpinyin_eucgb           163
gbpinyin_gb18030         163
gbpinyin_p936            163
gbpinyinnocs             26
gbpinyinnocs_eucgb       26
gbpinyinnocs_gb18030     26
gbpinyinnocs_cp936       26
```

For example, enter:

```
sp_configure 'default sortorder id', 26
```

SDK / Open Server version 15.0

For details about enhancements in 15.0 ESD features to Open Client/Open Server products, see the *New Features Open Server 15.0 and SDK 15.0 for Microsoft Windows, Linux, and UNIX*, or go to the Sybase information center, at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.dc20155_1500/html/newfesd/title.htm&toc=/com.sybase.help.os_15.0/toc.xml.

System changes

This section contains system changes that are not part of the sp_monitor enhancements. For those changes, see “System changes for sp_monitor” on page 10.

Changed functions

exportable keyword in *create function*

create function allows a new keyword, exportable, that determines whether the function can be forwarded to a remote server when used against a proxy table. exportable does not specify that a function must be exported, but only that it must be exported when the function is used with a proxy table. When possible, CIS includes the function in the SQL statement sent to the remote server. If the remote server is another Adaptive Server, the function must be defined in the user’s default database, so that the remote server can locate the function.

Syntax

```
create function[ owner ] function_name
( [ { @parameter_name [ AS ] parameter_datatype [ = default ] } [,...n ] ] )
returns return_datatype
[ exportable ]
[ with recompile] ]
[ as ]
begin
function_body
return scalar_expression
end
```

New functions

rand2

Description

Returns a random value between 0 and 1, which is generated using the specified seed value, and computed for each returned row when used in the select list.

Syntax

rand ([*integer*])

Parameter

integer is any integer (tinyint, smallint, or int) column name, variable, constant expression, or a combination of these.

Example

If there are n rows in table t, the following select statement returns n different random values.

```
select rand2 from t
------------
```

Usage

- Unlike rand, rand2 is computed for each returned row when it is used in the select list.
- rand2 is currently defined to work only with the the select list.
- For more information about the 32-bit pseudorandom integer generator, see the Usage section of rand, in the *Reference Manual: Blocks*.
- For general information about mathematical functions, see *Volume 1, Blocks*, of the *Reference Manual*.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute rand2.

See also

Datatypes Approximate numeric datatypes.

Changed stored procedures

sp_helptext

Description

New parameters for sp_helptext.

Syntax

sp_helptext [, "objname" , **linenum**, **numlines** , "**printoptions"**]

Parameters

linenum – specifies the starting line number from which the SQL text is generated.

numlines – specifies the number of lines for which to generate SQL text. If *printoptions* includes the term showsql, *numlines* specifies the number of lines of SQL text to display. If *printoptions* also includes the term context, *numlines* specifies the width of the context block surrounding *linenum*.

printoptions – a comma-separated list of terms specifying one or more properties for the output format, including:

- showsql – generates formatted SQL output for the compiled object.
- linenumbers – produces a line number for each line of SQL output.

- comments – produces the line numbers as a comment field, so that the generated SQL can be used, with additional edits, to recreate the compiled object.
- context – produces a context block of output around a specified starting line number. The width of this context block surrounding the line number is as specified by the parameter numlines. The default number of lines of context is 5.
- noparams – suppresses the automatically generated parameter information; use this option to produce only the portion of SQL output relevant to a compiled object.
- ddlgen – generates the SQL text as a minimal DDL script, prefacing output with use <database> and drop <object>.

Note ddlgen is for the use of sp_helptext only. Do not confuse it with the Sybase utility DDLGEN.

Examples

Example 1 To generate the formatted text for sp_help:

```
sp_helptext sp_help, NULL, NULL, "showsql"
```

Example 2 To generate the formatted SQL text for sp_help with line numbers:

```
sp_helptext sp_help, NULL, NULL,"showsql,linenumbers"
```

Example 3 To generate a context block of 7 lines, starting at line 25, with line numbers generated in a comment block:

```
sp_helptext sp_help, 25, 7,
       "showsql, linenumbers, comments, context"
```

Usage

- The object whose text you want to retrieve must reside in the database where the procedure is executed.
- If there is no text in syscomments, or if it is hidden by sp_hidetext, an error is reported, unless you request this context block output. In that case no error is raised, but a message reporting the missing text displays.
- If the compiled object contains a SQL select *, this statement is usually expanded on creation to reflect the entire column list of the table to which the select clause refers.

- SQL generated by ddlgen is minimal; for a more complete SQL script, use the Sybase ddlgen utility. The SQL generated by ddlgen may fail to create the compiled object if certain references to other objects, such as temporary tables, do not already exist when the generated script is executed.
- Using ddlgen and context together raises an error.

Changed monitoring tables

monProcess and monProcessWaits tables have a new column, ServerUserID, which maps to the spid's suser_id().This column allows filtering the result set by server user ID (suid).

System changes for *sp_monitor*

This section contains the system changes for the sp_monitor enhancement.

New *sp_monitor* configuration parameters

You must configure the monitoring table infrastructure before you can use the enhancements to sp_monitor.

The following examples are typical commands, performed to monitor deadlock events from the monDeadLock table.

```
sp_configure "deadlock pipe max messages",200
sp_monitor enable, deadlock
```

Note The configuration option enable monitoring must be ON for any kind of monitoring activity to take place.

Changed *sp_monitor* commands

sp_monitor 'enable' and 'disable'

Description

New parameters for sp_monitor enable and sp_monitor disable, which allow these procedures to take a monitoring entity type as an argument. This allows only those configuration options required to access data for that entity to be turned on or off. If no argument is provided, or all is provided, the default behavior is to turn all the required configuration options ON or OFF.

Syntax

sp_monitor enable [, "all" | '<*type*> [monitoring]']
sp_monitor disable [, "all" | '<*type*> [monitoring]']

Parameters

- *type* – connection, statement, event, procedure, deadlock, or procstack. To enable or disable monitoring of all the possible entities supported, use all. Table 1 on page 11 has information about the new monitoring types. See the table documenting all the monitoring types in the *Reference Manual: Stored Procedures*.

Table 1: Monitoring tables accessed by new monitoring types

Monitoring type	Tables accessed	Configuration option	Configuration option type
deadlock	monDeadlock	deadlock pipe max messages	Value
		deadlock pipe active	Boolean
procstack	monProcessProcedures	None	N/A

- monitoring – optional parameter that follows the name of each monitoring type.

Examples

Example 1

```
sp_monitor enable, connection
sp_monitor enable, 'statement monitoring'
...
sp_monitor disable, 'connection monitoring'
sp_monitor disable, 'all monitoring'
```

sp_monitor procstack

Description

Examines the execution context of a specified task, even within a deeply nested stored procedure execution.

Syntax

sp_monitor 'procstack', '*spid*' [, "*context*" [, '<*type*>']]

Parameters

spid – the server ID of the task analyzed.

context – the number of SQL lines of context around the line of text executed for each nested stored procedure. The default value is 5 lines of context for each procedure.

type – reserved for future use.

Examples

Example 1 Generates the procedural stack for the current spid executing sp_monitor.

```
1> sp_monitor procstack, '17'
2> go

No blocks were found for SPID 17
Procedure stack trace for SPID 17:


  Nesting DBName        OwnerName ObjectName          ObjectID   LineNumber
Blocked

   ------- -------------- --------- -------------------- ---------- -------
--- -------
2 sybsystemprocs NULL      sp_monitor_procstack  182288678        109
0
1 sybsystemprocs NULL      sp_monitor           2064723377       364
0

(2 rows affected)

 >>> SPID 17 [Nest: 2] Procedure 'sybsystemprocs..sp_monitor_procstack' at
line number 109: <<<

  CREATE PROCEDURE sp_monitor_procstack
Parameter_name Type    Length Prec Scale Param_order Mode
-------------- ------- ------ ---- ----- ----------- ----
@spid          int          4 NULL  NULL           1 in
@context       int          4 NULL  NULL           2 in
@type          varchar     10 NULL  NULL           3 in

(1 row affected)
104                           -- requesting, along with the locks held by
the task that is
105                           -- blocking this spid, if any.
106                           --
107                           set switch on 1202 with override, no_info
108
109 >>>                       select @blocking_spid = p.blocked
110                                , @linenum = linenum
111                                , @stmtnum = stmtnum
112                           from master.dbo.sysprocesses p
113                           where p.spid = @spid
```

```
114

QUERY PLAN FOR STATEMENT 52 (at line 131).

     STEP 1
       The type of query is SELECT.

       1 operator(s) under root


|ROOT:EMIT Operator
|
    |SCAN Operator
    |  FROM CACHE

 >>> SPID 17 [Nest: 1] Procedure 'sybsystemprocs..sp_monitor' at line number
364: <<<


  CREATE PROCEDURE sp_monitor
Parameter_name       Type    Length Prec Scale Param_order Mode
-------------------- ------- ------ ---- ----- ----------- ----
@entity              varchar     30 NULL  NULL           1 in
@dbname              varchar     30 NULL  NULL           2 in
@OrderBy_OR_Procname varchar     30 NULL  NULL           3 in
@option              varchar     30 NULL  NULL           4 in

(1 row affected)

    359                                  return 1
    360                              end
    361                   end
    362
    363                 -- Produce the procedural/stack trace for this spid.
    364 >>>               exec @rtnstatus = @monprocname @spid, @context,
@option
    365           end
    366
    367           else if @u_entity = "ARCHIVE"
    368           begin
    369                   -- Call the archival sproc for given monitoring
type and archive.

(return status = 0)
```

Example 2 `sp_monitor procstack, '14'`

```
go
SPID is involved in a blocking situation as follows:
spid   dbid     id   page   row   typestr           blocked
----   ----     --   ---    ---   -----------       -------
```

```
20     31514    3    1304    1   Ex_row-blk
14     31514    3    1304    1  Sh_row_request         20
Procedure stack trace for SPID 14:
.....
```

sp_monitor deadlock

Monitors and analyzes deadlock events from the monitoring table monDeadLock, and presents this data in human-readable format.

Syntax

```
sp_monitor deadlock [ [, filters ]
                                [, output_modes ] ]
```

To display deadlock events from either a monitoring table or an archive, enter:

```
sp_monitor deadlock
        [ [,NULL | '<deadlockID>' | '<forDate>']
          [, {  'verbose' | 'pagediag' }
           | { 'count by date'
               | 'count by application'
               | 'count by date, object'
             } ] ]
```

Parameters

filters – filters out deadlock events based on attributes including:

- *<deadlockID>* – the integer deadlock ID, entered as a string.
- *<forDate>* – the character representation of the date literal for which the deadlock events must be analyzed.

output_modes – allows the user to customize the output for each deadlock event. This parameter can be a combination of verbose and pagediag, separated by commas, or a summary output mode, which generates the frequency of deadlocks by object name, or application name, and so forth, rather than generating line item descriptions of each deadlock.

- verbose – produces details of the deadlock event: user, application names, and so forth.
- pagediag – produces page diagnostics that identify the type of page (data or index) in the deadlock event. This print option is only valid with the verbose output mode.
- The summary output modes give summary information on the frequency of deadlocks. They are 'count by date', 'count by application', 'count by date, object'.

Output examples

Example 1 A typical line item report:

```
> sp_monitor deadlock
```

```
**********Server: 'adit_125x_2k_bigbang' Deadlock ID 95 Dbname:
'tempdb' Resolve Time: Oct. 1 2006 1:12PM********
Deadlock ID 95 Spid 470 was waiting for 'exclusive intent' lock on dbid=2
object 'sysdepends' page=0
Deadlock ID 95 Spid 456 was holding 'exclusive table' lock on object
'sysdepends' page=0
TranName: '$ALTER TABLE #t2_alter_table ADD/DROP/MODIFY COLUMNS
ID=1055600629'

Deadlock ID 95 Spid 456 was waiting for 'exclusive table' lock on dbid=2
object 'syscomments' page=0
Deadlock ID 95 Spid 470 was holding 'exclusive table' lock on object
'syscomments' page=0. Command: 'CREATE TABLE', Holding
TranName: '$CREATE TABLE #t1_create_ta02004700014258171 allpages
executiontime'
```

Example 2 Generates the report for a particular deadlock event.

```
> sp_monitor deadlock, ID 97
**********Server: 'adit_125x_2k_bigbang' Deadlock ID 97 Dbname:
'tempdb' Resolve Time: Oct. 1 2006 1:12PM********
Deadlock ID 97 Spid 470 was waiting for 'exclusive intent' lock on dbid=2 object
'sysdepends' page=0
Deadlock ID 95 Spid 456 was holding 'exclusive table' lock on object
'sysdepends' page=0
TranName: '$ALTER TABLE #t2_alter_table ADD/DROP/MODIFY COLUMNS
ID=1087600743'

Deadlock ID 97 Spid 456 was waiting for 'exclusive table' lock on dbid=2 object
'syscomments' page=0
Deadlock ID 97 Spid 470 was holding 'exclusive table' lock on object
'syscomments' page=0. Command: 'CREATE TABLE', Holding
TranName: '$CREATE TABLE #t1_create_ta02004700014258171 allpages executiontime'
```

Example 3 Tracking the code causing a deadlock:

```
sp_monitor deadlock, '120', "verbose, pagediag"
```

verbose produces the SQL text, with line numbers, of procedures involved in the deadlock. pagediag analyzes the pages, if any, involved in the deadlock.

Usage

Use sp_configure to manually set monitoring table infrastructure:

```
sp_configure 'deadlock pipe max messages', 200
```

sp_monitor archive, report

Description

Two commands in the sp_monitor syntax, archive and report, support archiving monitoring data to user tables, and reporting from the archived data, for offline analysis.

Syntax

```
sp_monitor 'archive [using prefix=<string>]', {<monitoring type>}
sp_monitor 'report [using prefix=<string>]', {<monitoring type >}
          [,<options supported for monitoring_type>]
```

Example 1 Both archive and report commands include an optional using clause, that introduces the properties of the archive table. Data is archived into a table in the database from which the procedure is executed. The archive tables are assigned the same names as the original monitoring tables. Example 1 archives data into a table named monDeadLock in the current database, mondb.

```
use mondb
go
sp_monitor archive, deadlock
go
```

Example 2 Contains the subclause USING prefix=daily, to archive monitoring data in a table named daily_monDeadLock, in the database mondb:

```
mondb..sp_monitor "archive USING prefix=daily_", deadlock
```

Example 3 This use of reports on all deadlock events in the archived table monDeadLock, in the current database, mondb:

```
sp_monitor "report", deadlock
```

If the current database is the master database, this command is equivalent to the next example, in that both commands process a table using the default name: monDeadLock.

This command also processes a table using the default name:

```
sp_monitor deadlock
```

Example 4 The report command reports from an archived table, and passes any reporting arguments supported by deadlock. The examples below process a table with the default name monDeadLock, and generate a report on deadlock events archived in a table called daily_monDeadlock, using various reporting modes.

```
# Report for deadlock ID 10
mondb..sp_monitor "report USING prefix=daily_",
deadlock, '10'

# Report for deadlock ID 10, in verbose mode,
```

```
# including page diagnostics.
mondb..sp_monitor "report USING prefix=daily_",
deadlock, '10',
"verbose,pagediag"

# Report in verbose mode for all deadlock events
# on a given date
mondb..sp_monitor "report USING prefix=daily_"
deadlock, "Mar 16 2006", "verbose"

# Report a frequency count of deadlocks by date
# from the archived data.
mondb..sp_monitor "report USING prefix=daily_",
deadlock, NULL,'count by date'
```

Example 6 These examples show how to send monitoring data to an archive carrying the default name, and processing it to identify deadlocks for a specified date, group it by various attributes, and so forth.

```
1> tempdb..sp_monitor archive, deadlock
2> go

Created new archive 'tempdb.dbo.monDeadLock'.
Archived 62 rows to archive table
'tempdb.dbo.monDeadLock' with timestamp 'Jul 11
2006 5:24AM'
(return status = 0)

1> tempdb..sp_monitor report, deadlock, NULL,
'count by object'
2> go

Frequency of deadlocks, by 'DBName, ObjectName'
from 'tempdb.dbo.monDeadLock':

DBName     ObjectName   Frequency
---------  -----------  -----------
pubs2      mycols            12
pubs2      myobjs            12
pubs2      mycols_apl        12
sa_tempdb  sysdepends        10
sa_tempdb  syscomments       10
tempdb     sysdepends        3
tempdb     syscomments       3

(1 row affected)
(return status = 0)

1> tempdb..sp_monitor report, deadlock, NULL, 'count by date'
2> go
```

```
Frequency of deadlocks, by 'ResolveDate' from
'tempdb.dbo.monDeadLock':

ResolveDate       Frequency
------------      -----------
Jul 11 2006             31

(1 row affected)
(return status = 0)

1> tempdb..sp_monitor report, deadlock,
'Jul 11 2006', 'count by object'
2> go

Frequency of deadlocks, by 'DBName, ObjectName'
from 'tempdb.dbo.monDeadLock':

DBName     ObjectName     Frequency

---------   ----------    -----------
pubs2       mycols            12
pubs2       myobjs           12
pubs2       mycols_apl       12
sa_tempdb   sysdepends       10
sa_tempdb   syscomments      10
tempdb      sysdepends        3
tempdb      syscomments       3

(1 row affected)
(return status = 0)
```

The examples above demonstrate how the existing *filter* and *output_modes* parameters, documented in sp_monitor, the *Reference Manual: Stored Procedures*, can process data from an archive exactly as they process data directly from the monitoring table.

Usage

- Choose default archive names based on the archived monitoring table.
- You can change the default archive name with a user-supplied prefix.
- Use report to re-create the deadlock event trace from archived data.
-

  If you attempt to archive the data to a table without enabling select into in the database where you create the archive table, an error similar to the following results:

  ```
  mondb..sp_monitor archive, deadlock
  -------------------------------
  Cannot use database 'mondb' as an archive database
  for monitoring data as it does not have the select
  ```

```
    into/bulkcopy/pllsort' set.
    return status = 1
```

- You cannot use archive in system databases: master, model, sybsystemdb, sybsystemprocs, sybsecurity.
- You cannot use proxy databases as archive databases.
- Sybase recommends that you do not use temporary tables as archive repositories: they are deleted when the session generating the archive disconnects from the server.

Permissions

For sp_monitor, mon_role and sa_role privilege are required to archive data from the monitoring tables, and select privilege is required to access data on the archive table.

sp_monitor help

Description

Provides command-specific examples for other enhanced commands. The syntax of this command has not changed.

Syntax

sp_monitor help [, { command | monitoring_type }]

Parameters

- command – name of a command, such as enable or disable.
- monitoring_type – monitoring type such as deadlock or connection. For a complete table of these types see Table 1 on page 11.

Examples

```
sp_monitor help, enable
----------------------
Usage: sp_monitor 'enable' [, '<type> [monitoring]'
Examples: sp_monitor 'enable'
Valid monitoring <type> is one of : 'enable' | 'disable'
| 'connection' | 'procedure' | 'statement' | event' |
'deadlock' | 'procstack'

-- Enable monitoring for a specific type, to monitor
statements.
sp_monitor enable, statement
sp_monitor enable, 'statement monitoring'

-- Enable monitoring for connection monitoring
sp_monitor enable, connection
sp_monitor enable, 'connection monitoring'

-- Enable monitoring for all types
sp_monitor enable
sp_monitor enable, 'all'
sp_monitor enable, 'all monitoring'
```

```
sp_monitor help, procstack
------------
Usage: sp_monitor 'procstack' [, '<spid>']
[, '<contextbloc>' ]

-- Monitor a given spid (that may possibly be in a hung
state), and if possible
-- generate a back trace of the procedures it is
currently executing.
sp_monitor 'procstack', '<spid>'

-- To generate a procstack of your current connection,
with
a context block of 10 lines, do:
declare @spid_str varchar(5)
select @spid_str = convert(varchar, @@spid) exec
sp_monitor 'procstack', @spid_str, '10'
```

sp_monitor list

Description — Lists the currently enabled entities and necessary configuration options enabled for monitoring.

Syntax — sp_monitor list

Sample output

```
1> sp_monitor list
2> go

Monitoring is currently active for the following types:

 mon_type
---------------------
connection monitoring
event monitoring
procedure monitoring
procstack monitoring
statement monitoring

(5 rows affected)

Monitoring types that are enabled but require some additional configuration
options to be turned ON:


 mon_type      config_name                run_value    cfg_value type

 -------- -------------------------       ---------   --------- -------
deadlock      deadlock pipe max messages         0         0 dynamic
```

```
(1 row affected)

Monitoring related configuration options found to be enabled via sp_monitor:


 mon_type   config_name          run_value enabled              type
---------- -------------------- --------- ------------------- -------
deadlock   enable monitoring            1 Sep  3 2007  5:25PM dynamic
deadlock   deadlock pipe active         1 Sep  3 2007  5:25PM dynamic

(2 rows affected)

Monitoring related configuration options found to be enabled via
sp_configure:

config_name                   run_value cfg_value   type    mon_type

 ---------------------------- --------- --------- ------- ----------------
-------------------

 max SQL text monitored          2048     2048 static  connection, statement
wait event timing                    1   1
dynamicconnection,deadlock,event,statement

 per object statistics active         1         1 dynamic
connection,procedure,statement
SQL batch capture                   1         1 dynamic connection,statement
process wait events                   1         1 dynamic event
statement pipe active                 1         1 dynamic procedure
statement statistics active          1         1 dynamic procedure,statement
statement pipe max messages     100000    100000 dynamic procedure,statement

(8 rows affected)
(return status = 0)
```

Usage

- Use list to check whether the desired monitoring type is activated, or whether some configuration option is still missing.
- Reports the configuration options that were set by sp_monitor, versus those options that have been set using configuration files or sp_configure.