SYBASE®

An SAP® Company

New Features Guide

# Adaptive Server® Enterprise

# 15.7 SP60

DOCUMENT ID: DC00641-01-157060-01
LAST REVISED: December 2013

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

# Contents

# Certicom Replacement

Certicom software, which provides cryptography services for securing storage and transmission of sensitive information, is no longer supported by SAP® Sybase products. These services have been replaced by alternate providers, as indicated in the documentation for each SAP Sybase product.

OpenSSL is the Adaptive Server® supported provider on all platforms. For information about OpenSSL, see the OpenSSL Web site at *http://www.openSSL.org"*. Changes due to the provider replacement are:

### *Certificate Management*

These certificate utilities are no longer supported:

- `certreq`
- `certauth`
- `certpk12`

As a replacement, Adaptive Server includes the `openssl` open source utility in:

- (UNIX) `$SYBASE/$SYBASE_OCS/bin/openssl`
- (Windows) `%SYBASE%\%SYBASE_OCS%\bin\openssl`

Use `openssl` to accomplish all certificate management tasks previously implemented by `certreq`, `certauth`, and `certpk12`. For information about the use of the `openssl` utility, go to *http://www.openssl.org/docs/apps/openssl.html*.

### *Enabling FIPS Encryption*

In previous releases, enabling the **FIPS login password encryption** parameter specified the use of FIPS 140-2 compliant cryptographic module for the encryption of passwords in transmission, in memory, and on disk. For Adaptive Server 15.7 SP60, enabling this parameter specifies that the FIPS 140-2 compliant cryptographic module is used for all encryption related operations.

Also in previous releases, FIPS Encryption was turned on by default. However, for Adaptive Server 15.7 SP60 it must be explicitly enabled. Client libraries must also enable FIPS to complete FIPS configuration.

**Note:** FIPS Certification is not supported for IBM AIX and Linux on POWER platforms.

### *Certificate Generation*

OpenSSL in FIPS mode is strictly controlled by OpenSSL security. This means that some certificates that worked with the Certicom FIPS module may no longer work using OpenSSL. In particular, the use of MD5 algorithm is not FIPS 140-2 compliant. Old certificates using this algorithm must be replaced in order to enable the **FIPS login password encryption** parameter.

When generating a FIPS compliant certificate, FIPS 140-2 compliant algorithms must be used. Private keys must be in pkcs8 format and encrypted with an OpenSSL FIPS 140-2 compliant algorithm.  To determine what algorithm is used to encrypt a private key, enter the following command:

```
openssl asn1parse -in <Encrypted Key Filename> -inform PEM
```

To convert the key to the proper format use the following command:

```
openssl pkcs8 -in <Non-FIPS compliant Encrypted Key Filename>
-topk8 -out <FIPS compliant Encrypted Key Filename>
-v1 PBE-SHA1-3DES
```

### *Digital Signature RSA Encryption Algorithms*

If RSA encryption algorithms are used for the digital signature, the RSA key size must be at least 1024 bit.

### *Cipher Support*

When configuring FIPS cipher suites, the supported cipher suites have changed.  These are the supported cipher suites for FIPS:

- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA

# Transaction Log Space Management

Adaptive Server introduces new functionality to analyze and free transaction log space.

Adaptive Server provides a single transaction log segment per database. Space can be added to a log segment or removed from a log segment. However, at any given point, there is limited space in a log segment.

Whenever the database client applications perform any data manipulation language (DML) operations on the data, Adaptive Server produces log records that consume space in the transaction log. Typically there are several clients performing DMLs concurrently on the database and log records are appended to the log whenever user log caches (ULCs) for individual clients are full or in some other conditions such as when a data page is changed by multiple open transactions. Log records of several transactions are therefore typically interleaved in the transaction log space.

Removing transactions from the transaction log to free the log space can be done using **dump transaction**. However, there different scenarios that can cause the transaction log can grow in such a way that the **dump transaction** command is not able to free space. In these situations, the log consumes space to such an extent that it affects the continuous availability of the database system for any write operations. The scenarios include:

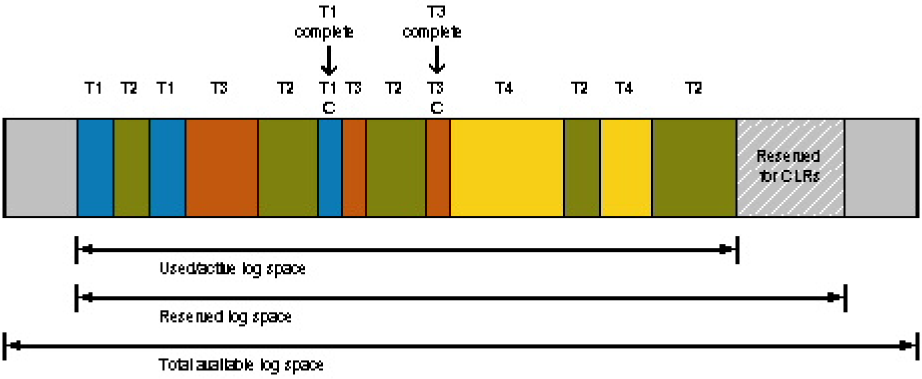
- Transactions are entered into the server but not committed. In this situation the log cannot be truncated because there is an active transaction.
- Replication Server is slow in reading the log which prevents truncating the log.
- A dump transaction has not been performed for a long period of time. Periodically dumping transactions can keep the size of the reserved space in the log limited and ensure that there is free space available in the log which allows the space freed after dump transaction to be reused for further logging activity.

You can use the **loginfo** function to evaluate how the space of a transaction log is used and determine the type of actions possible to free log space. The **sp_thresholdaction** procedure can be used to free log space in the transaction log if the available free space falls below a preconfigured threshold. The recommended action is to define a trigger that will execute **dump transaction** once the log fall below the threshold. However, the **dump transaction** command cannot truncate the portion of the log beginning from of the oldest incomplete or active transaction in the log, since this portion is needed for recovery of the database. In this case, the oldest transactions can be aborted, depending on circumstances.

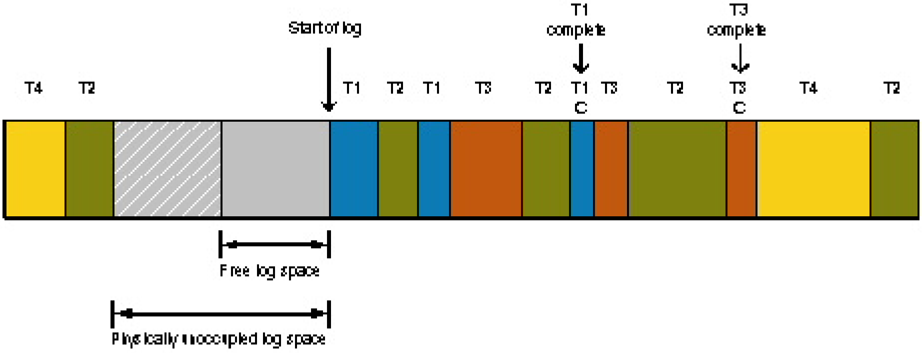
## Transaction Log Space

The transaction log contains sufficient information about each transaction to ensure that it can be recovered. This transaction information can be interleaved within the log.

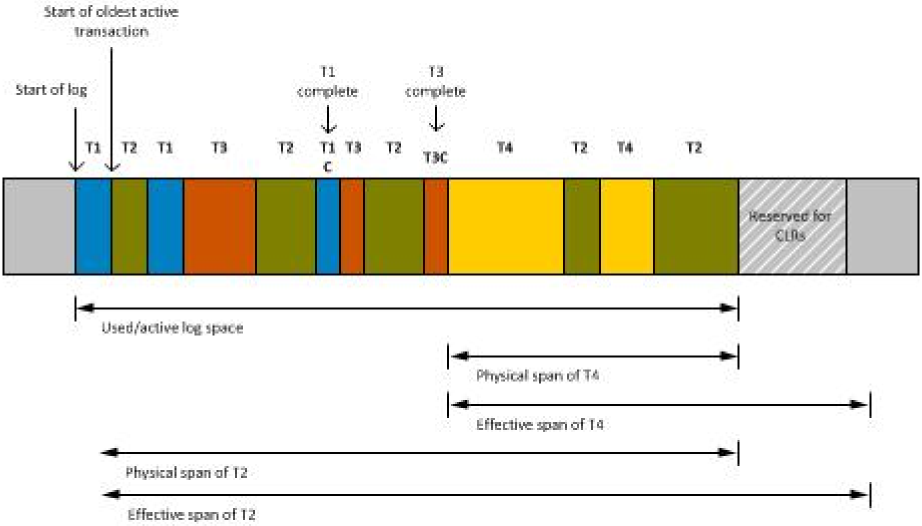
This transaction log shows the space allocation of the transaction log.



If a total allocated log segment is considered a contiguous logical space, then additional log records are appended to the end of the log. When there is no more space available at the end of the log, portions of the log that have been freed in the beginning of allocated space will be used to add records. This use of the freed portion of log can introduce a circular or wrapped around form within the allocated space.



The span of a completed transaction in the log is the portion of a log between begin log record of the transaction in the log and end log record of the transaction. The actual logging done by a transaction could be very minimal, however due to intermediate interleaved log records by other transaction, this span in terms of transaction log space could be considerable.



As transaction log space allocation is done in terms of pages, the span of an active transaction or incomplete transaction is also defined in terms of the number of log pages. There can be many active transactions at any point in the log. The portion of the transaction log occupied by span of an active transaction cannot be truncated with **dump transaction**. The total portion of the log which cannot be truncated with **dump transaction** is equal to span of the oldest active transaction

# loginfo

The **loginfo** function has been extended in Adaptive Server to further support managing transaction log space.

## Syntax

```
loginfo (dbid | dbname, option]
```

```
loginfo (dbid | dbname, option, option1]
```

## Parameters

- ***dbid* –** is the database ID.
- ***dbname* –** is the database name.
- ***option* –** is the specific information you need about the transaction log.

  These options have been added in Adaptive Server 15.7 SP110:

  - **oldest_active_transaction_pct** – returns a number from 0 to 100 indicating the span of the oldest active transaction in percentage of total log space.

- **oldest_active_transaction_spid** – returns the spid of the session having the oldest active transaction in the log of the Adaptive Server.
- **oldest_active_transaction_page** – returns the logical page number of start of oldest active transaction in the log. Returns 0 if there is no active transaction.
- **oldest_active_transaction_date** – returns the start time of oldest active transaction. Returns binary(8) number which needs to be converted to date as shown in the example below:

  ```
  select (convert(datetime, convert(binary(8),
      loginfo(4, 'oldest_active_transaction_date')), 109))
  ```
- **is_stp_blocking_dump** – returns 1 if there is a secondary truncation point before the start of the oldest active transaction, otherwise, returns 0.
- **stp_span_pct** – returns a number from 0 to 100 indicating the span of secondary truncation point to the end of log with respect to total log space.
- **can_free_using_dump_tran** – returns a number from 0 to 100 indicating the span of transaction log which can be truncated with the **dump transaction** command without having to abort oldest active transaction. If there is a secondary truncation point before the start of the oldest active transaction, then this is the span in the log (in percent) between the start of the log (first log page) and the secondary truncation point. If the secondary truncation point is not before the oldest active transaction, then this is the span in the log (in percent) between the start of the log (first log page) and start of the oldest active transaction.
- **is_dump_in_progress** – returns 1 if **dump transaction** command is in progress, returns 0 if no dump command is in progress.
- **database_has_active_transaction** – returns 0 if there are no active transactions in the log. Returns 1 if there is an active transaction in the log.
- **xactspanbyspid** – This option is to be used only with the third parameter, which is the SPID of the task. Returns the transaction span if the SPID has an active transaction in the log. Returns 0 otherwise.

These options are available in Adaptive Server 15.7 SP100 and later:

- **help** – shows a message with the different options.
- **first_page** – returns the page number of the first log page.
- **root_page** – returns the page number of the last log page.
- **stp_page** – returns the page number of the secondary truncation point (STP), if it exists. The secondary truncation point (or STP) is the point in the log of the oldest transaction yet to be processed for replication. The transaction may or may not be active. In cases where the transaction is no longer active, the STP by definition precedes the oldest active transaction.
- **checkpoint_page** – returns the page number in the log that contains the most recent checkpoint log record.
- **checkpoint_marker** – returns the record ID (RID) in the log that contains the most recent checkpoint log record.

- **checkpoint_date** – returns the date of the most recent checkpoint log record.
- **oldest_transaction_page** – returns the page number in the log on which the oldest active transaction at the time of the most recent checkpoint, started. If there was no active transaction at the time of the most recent checkpoint, **oldest_transaction_page** returns the same value as **checkpoint_page**.
- **oldest_transaction_marker** – returns the RID (page number and row ID) in the log on which the oldest active transaction at the time of the most recent checkpoint, started. If there was no active transaction at the time of the most recent checkpoint, **oldest_transaction_marker** returns the same value as **checkpoint_marker**.
- **oldest_transaction_date** – is the at which the oldest active transaction started.
- **until_time_date** – is the latest time that could be encapsulated in the dump that is usable by the **until_time** clause of **load transaction**.
- **until_time_page** – is the log page on which the log record associated with **until_time_date** resides.
- **until_instant_marker** – is the RID (page number and row ID) of the log record associated with **until_time_date**.
- **total_pages** – is the total number of log pages in the log chain, from **first_page** to **root_page**.
- **stp_pages** – the total number of log pages between the STP and the oldest active transaction.
- **active_pages** – the total number of pages between the oldest transaction at the time of the most recent checkpoint, and the end of the log.
- **inactive_pages** – the total number of log pages between **first_page** and either **stp_page** or **oldest_transaction**, whichever comes first. This is the number of log pages that will be truncated by the **dump transaction** command.

**Note:** For a Mixed Log Data (MLD) database, this function returns a value equivalent to 0. The new options for this function are not supported or meant to be used for MLD databases.

## Examples

- **Example 1 –** Shows how to display transaction log information.

```
select loginfo(dbid, 'database_has_active_transaction'),
       loginfo(dbid, 'oldest_active_transaction_pct'),
       loginfo(dbid, 'oldest_active_transaction_spid'),
       loginfo(dbid, 'can_free_using_dump_tran'),
       loginfo(dbid, 'is_stp_blocking_dump'),
       loginfo(dbid, 'stp_span_pct')
```

```
has_act_tran OAtran_spid Act_log_portion_pct dump_tran_free_pct is_stp_blocking stp_span_pct log_occupied_pct
------------ ------------ ------------------- ------------------ --------------- ------------ ---------------
     1            14               17                  7                 0            25            32
```

The function returns the transaction log information:

  - 1 active transaction
  - 14 is the SPID of the oldest transaction
  - 17 percent of the log that is occupied by an active transaction
  - 7 percent of the transaction log that can be freed by using the dump transaction command
  - 0 blocking secondary truncation points
  - 25 percent of the log that is occupied by the span of the secondary truncation point
  - 32 percent of the log that is occupied

- **Example 2 –** Returns the amount of log space that is spanned for a particular transaction.

  ```
  select loginfo(dbid, ‘xactspanbyspid’, spid)
  spid      log_span_pct
  ----------------------
    15           2
  ```

### Permissions

The user must have sa_role to execute **loginfo**.

## sp_xact_loginfo

**sp_xact_loginfo** provides the span of oldest active transaction in terms of percentage of total log space.

### Syntax

```
sp_xact_loginfo dbid [, vcharparam1] [, vcharparam2]
[, intparam1] [, intparam2]['  span_pct] [, startpage]
[, xact_spid] [, starttime] [, firstlog_page] [, stp_page]
[, stp_pages] [, stp_blocking] [, canfree_without_abort_pct]
[, dump_in_progress] [, activexact] [, errorcode]
```

### Parameters

- **dbid –** is the database ID.
- **vcharparam1 –** varchar parameter indicating the mode. If **oldestactive**, the output parameter values are indicative of oldest active transaction. If **xactspanbyspid**, then output parameter values reflect values of active transaction for given spid.
- **vcharparam2 –** reserved for future use. Provide NULL as a value.
- **intparam1 –** integer parameter1 (SPID if *vcharparam1* = **xactspanbyspid**)
- **intparam2 –** integer parameter2
- **span_pct –** a value from 0 to 100. Indicates the span of transaction in percentage of total log space based on value of *vcharparam1*(output parameter).

- **startpage –** page number that is the start of the active transaction in the log based on value of vcharparam1. This page will hold the begin transaction log record of the active transaction.
- **xact_spid –** server process ID of the client having the active transaction based on *vcharparam1*.
- **starttime –** start time of active transaction based on *vcharparam1.*
- **firstlog_page –** server process ID of the client having active transaction based on *vcharparam1*.
- **stp_page –** secondary truncation point logical page number in the log. Returns -1 if replication is not active.
- **stp_pages –** returns the total number of log pages between the secondary truncation point and the oldest active transaction. Returns 0 if:
  - replication is not active
  - there is no active transaction in the log
  - there is no secondary truncation point before oldest active transaction
- **stp_blocking –** value of 0 or 1. 1 indicates that the secondary truncation checkpoint will block some portion for truncation beyond oldest active transaction span. Meaning that secondary truncation point is in between the start of the log and the start of oldest active transaction and replication agent must catch up. 0 indicates that aborting the oldest active transaction will free transaction log space without the secondary checkpoint blocking the abort.
- **canfree_without_abort_pct –** value from 0 to 100. Indicates the difference between **startlogpagenum** and **startxactpagenum** in terms of percentage of total log space. This portion can be truncated with the dump transaction command without aborting the oldest active transaction.
- **dump_in_progress –** returns1 if the dump transaction command is in progress, returns 0 if no dump command is in progress. Values of output parameters **firstlog_page** and **canfree_without_abort_pct** are not reliable. (output parameter).
- **activexact –** Boolean flag indicating that there are active transactions in the log.
- **errorcode –**
  - 0 there are no errors
  - 1 insufficient permission to execute
  - 2 error in opening dbtable. This could be due to various reasons including the dbid or database name given does not exist.
  - 3 cannot start xls session for log scan.
  - 4 there are no open transaction in the log against this database

**Note:** For a Mixed Log Data (MLD) database, this procedure returns values equivalent to 0 in output parameters. This procedure is not supported or meant to be used for MLD databases.

## Automating Transaction Log Management

Use cases to automate the management of the transaction log.

You can use **sp_thresholdaction** to identify the oldest active transaction.

You can use **sp_xact_loginfo** to monitor the longest running transaction per database or abort the oldest active transaction based on conditional criteria.

### Rescue Scenario Use Case

**sp_thresholdaction** can be used to identify the oldest active transaction and decide on action based on the information returned.

You can truncate the log to free up the space or abort the oldest active transaction or both based on the defined criteria. This use case assumes that the oldest active transaction span needs to be watched or limited only when free space in log segment falls beyond threshold.

```
create procedure sp_logthresholdaction
        @dbname         varchar(30),
        @segmentname    varchar(30),
        @space_left     int,
        @status         int
as
declare
        @oats_span_pct           int,
        @dbid                    int,
        @at_startpage            bigint,
        @firstlog_page           bigint,
        @canfree_withoutabort    int,
        @stp_blocking            int,
        @stp_pagebig             int,
        @dump_in_progress        int,
                @oat_spid               int,
                @oat_starttime          datetime,
                @activexact             int,
                @free_with_dumptran_pct int,
                @errorcode              int,
                @xactspan_limit         int,
                @space_left_pct         int,
                @dumptrancmd            varchar(256),
                @dumpdevice             varchar(128),
                @killcmd                varchar(128)

        select @dbid = db_id(@dbname)
        select @dumpdevice = “/ferrum_dev1/db1.tran.dmp1”

        select @free_with_dumptran_pct = 5
/*
** attempt dump tran only if it can free at
** least 5 percent of log space
*/
        select @xactspan_limit = 20
/*
```

```
** optionally also kill oldest active transaction even after
** dump tran if its exceeds 20 percent in the log
*/

    select @space_left_pct = logspace_pagestopct(@dbid, @space_left, 0)
    print “Space left in log segment “ + @space_left_pct + “ percent.”

    select @dump_in_progress = 1
    while (@dump_in_progress> 0)
        begin    -- {
            exec sp_xact_loginfo@dbid,
            “oldestactive”,
            NULL,
            0,
            0,
            @span_pct = @oats_span_pct output,
            @startpage = @oat_startpage output,
            @xact_spid = @oat_spid output,
            @starttime = @oat_starttime output,
            @firstlog_page = @firstlog_page output,
            @stp_page = @stp_page output,
            @stp_blocking = @stp_blocking output,
            @canfree_without_abort_pct = @free_with_dumptran_pct output,
            @dump_in_progress = @dump_in_progress output,
            @activexact = @activexact output,
            @errorcode = @errorcode output

if (@dump_in_progress> 0)
begin
    sleep 30
    continue
end
select @killcmd = “kill “ + @xact_spid + “ with status_only”

if (@canfree_withoutabort>@free_with_dumptran)
begin
     select @dumptrancmd = “dump tran ” + @dbname + “ on “ + @dumpdevice
     exec(@dumptrancmd)
/*
** Optionally, also abort oldest active transaction.
*/
        if ((@stp_blocking = 0) and
        (@oats_span_pct> @xactspan_limit))
        then
/*
** Some diagnostic information can be printed or warning actions
** can be executed here before aborting the transaction.
*/
            exec(@killcmd)
        end
else
/*
** Some diagnostic information can be printed or warning actions
** can be executed here before aborting the transaction.
*/
    exec(@killcmd)
    end
    end -- }
```

### Monitoring Use Case

**sp_xact_loginfo** can be used for periodically monitoring the longest running transaction per database.

For example a stored procedure can be formed around **sp_xact_loginfo** in which it populates the tables with the oldest active transaction information and populates a user defined table. The execution of this stored procedure can be periodic, at desired frequency through job scheduler.

```
/*
** Stored procedure assumes presence of a pre-created table
** for monitoring oldest active transactions with following
** table definition:
**
**    create table oldest_active_xact(
**           dbid                     int,
**           oldestactivexact_span    int,
**           startxactpagenum         int,
**           spid                     int,
**           xactstarttime            varchar(27),
**           startlogpagenum          int,
**           stppage                  bigint,
**           sec_truncpoint_block     int,
**           can_free_wo_kill         int,
**           dump_in_progress         int,
**           nolog                    int,
**           username                 varchar(30) null)
*/
create procedure sp_record_oldestactivexact
@dbname    varchar(30)
as
declare     @dbid int,
            @oats_span_pct int,
            @oat_startpage bigint,
            @firstlog_page bigint,
            @canfree_withoutabort int,
            @stp_blocking int,
            @stp_page bigint,
            @dump_in_progress int,
            @oat_spid int,
            @oat_starttime varchar(27),
            @activexact int,
            @free_with_dumptran_pct int,
            @errorcode int,
            @username varchar(30)

select @dbid = db_id(@dbname)

exec sp_xact_loginfo @dbid,
            'oldestactive',
            NULL,
            0,
            0,
            @span_pct = @oats_span_pct output,
            @startpage = @oat_startpage output,
            @xact_spid = @oat_spid output,
            @starttime = @oat_starttime output,
            @firstlog_page = @firstlog_page output,
            @stp_page = @stp_page output,
            @stp_blocking = @stp_blocking output,
            @canfree_without_abort_pct = @free_with_dumptran_pct output,
            @dump_in_progress = @dump_in_progress output,
            @activexact = @activexact output,
            @errorcode = @errorcode output

if (@activexact = 1)
    begin
        print "activexact is true"
        select @username = suser_name(sysproc.uid)
                from master..systransactions systran,
                master..sysprocesses sysproc
                      where systran.spid = @oat_spid
                and systran.spid = sysproc.spid
        insert into oldest_active_xact
            values(
                @dbid,
                @oats_span_pct,
                @oat_startpage,
                @oat_spid,
                @oat_starttime,
                @firstlog_page,
                @stp_page,
                @stp_blocking,
                @free_with_dumptran_pct,
                @dump_in_progress,
                @activexact,@username)
    end
else
    begin
        print "activexact is false"
        insert into oldest_active_xact values(
                @dbid,
                @oats_span_pct,
                @oat_startpage,
                @oat_spid,getdate(),
                @firstlog_page,
                @stp_page,
                @stp_blocking,
                @free_with_dumptran_pct,
                @dump_in_progress,
                @activexact,NULL)
end
```

### Monitoring and Control Use Case

In addition to monitoring, the action of aborting the oldest active transaction based on conditional criteria can also be implemented in **sp_xact_loginfo** which is run periodically through job scheduler.

```
/*
** Stored procedure assumes presence of a pre-created table for
monitoring
** oldest active transactions with following table definition:
**
**    create table oldest_active_xact(datetimetime_of_recording,
**           dbid                     int,
**           oldestactivexact_span    int,
**           spid                     int,
**           username                 varchar(30),
**           startxactpagenum         int,
**           startlogpagenum          int,
**           xactstarttime            datetime,
**           can_free_wo_kill         int,
**           sec_truncpoint_block     int,
**           nolog                    int,
**           action_taken             varchar(30))
**           lock datarows
*/
create procedure sp_control_oldestactivexact
           @dbname    varchar(30)
as
declare    @oats_span_pct           int,
           @dbid                    int,
           @at_startpage            bigint,
           @firstlog_page           bigint,
           @canfree_withoutabort    int,
           @stp_blocking            int,
           @stp_pagebig             int,
           @dump_in_progress        int,
           @oat_spid                int,
           @oat_starttime           datetime,
           @activexact              int,
           @free_with_dumptran_pct  int,
           @errorcode               int,
           @username                varchar(30),
           @action_taken            varchar(30),
           @xact_maxspan_pct        int,
           @killcmd                 varchar(128)

    select @dbid = db_id(@dbname)
    select @xact_maxspan_pct = 20
    select @action_taken = “none”

exec sp_xact_loginfo @dbid,
        “oldestactive”,
        NULL,
        0,
        0,
@span_pct = @oats_span_pct output,
@startpage = @oat_startpage output,
@xact_spid = @oat_spid output,
@starttime = @oat_starttime output,
@firstlog_page = @firstlog_page output,
@stp_page = @stp_page output,
@stp_blocking = @stp_blocking output,
@canfree_without_abort_pct = @free_with_dumptran_pct output,
@dump_in_progress = @dump_in_progress output,
@activexact = @activexact output,
@errorcode = @errorcode output

    select @killcmd = “kill “ + @oldesactive_spid + “ with status_only”

    if (@nolog == 0)
    then
select @username = suser_name(systran.suid)
from master..systransactionssystran where systran.spid
=@oldestactive_spid
if (@oats_span_pct> @xact_maxspan_pct)
begin
    exec(@killcmd)
    select @action_taken = “transaction abort”
end
        insert into oldest_active_xact values(getdate(), @dbid,
@oats_span_pct, @oat_spid, @username, @oat_page, @firstlog_page,
@free_with_dumptran_pct, @stp_blocking, @activexact, @action_taken)
    else
        /*
        ** Just to cover possibility of no active transactions which have
        ** generated any log.
        */
        insert into oldest_active_xact values(getdate(), @dbid,
                    0, 0, NULL, 0, 0, 0, 0, 1, @action_taken)
    end
```

## Analyzing and Managing Transaction Log Space

Use the **loginfo** function to view and free transaction log space.

The system administrator can use the **loginfo** function to evaluate how the space of a transaction log is used and determine the type of actions possible to free space.

This example uses **loginfo** to show the transaction log at a particular point in time:

```
=====================================
select loginfo(dbid, 'database_has_active_transaction') as has_act_tran,
       loginfo(dbid, 'oldest_active_transaction_pct') as Act_log_portion_pct,
       loginfo(dbid, 'oldest_active_transaction_spid') as OA_tran_spid,
       loginfo(dbid, 'can_free_using_dump_tran') as dump_tran_free_pct,
       loginfo(dbid, 'is_stp_blocking_dump') as is_stp_blocking,
       loginfo(dbid, 'stp_span_pct') as stp_span_pct
```

```
has_act_tran OAtran_spid Act_log_portion_pct dump_tran_free_pct is_stp_blocking stp_span_pct log_occupied_pct
------------ ------------ ------------------- ------------------ --------------- ------------ ---------------
      1           19               38                  7                 0              45            52
```

```
(return status = 0)
```

This shows:

- `has_act_tran` = 1, indicates that the database currently has one active transaction.
- `OA_tran_spid` = 19, indicates that the system process ID of the oldest active transaction in the database is 19.
- `Act_log_portion_pct` = 38, indicates that 38 percent of the log space is occupied by the oldest active transaction.
- `dump_tran_free_pct` = 7, indicates that 7 percent of the transaction log that can freed using **dump transaction**.
- `is_stp_blocking` = 0, indicates that there is no secondary truncation point preventing the use of **dump transaction** to free space.
- `stp_span_pct = 45`, indicates that there is a secondary truncation point spanning 45 percent of the transaction log.
- `log_occupied_pct` = 52, indicates that 52 percent of the total log space is currently occupied.

The available actions are:

1. The first step can be to use **dump transaction** to free the transaction log of the 7 percent shown by `dump_tran_free_pct = 7`. After freeing the space using **dump transaction**, the output of the same query shows:

```
has_act_tran OAtran_spid Act_log_portion_pct dump_tran_free_pct is_stp_blocking stp_span_pct log_occupied_pct
------------ ------------ ------------------- ------------------ --------------- ------------ ---------------
      1           19                38                  0                 1             45            45

(return status = 0)
```

2. At this stage, `Act_log_portion_pct = 38`, indicates that 38 percent of the log space is occupied by the transaction with the system process ID of 19. You can wait for system process 19 to complete, or abort the transaction.

   If you decide to abort the transaction using the **kill** command (with or without status only option) as a measure to rescue the log, re-issuing the same query shows:

```
has_act_tran OAtran_spid Act_log_portion_pct dump_tran_free_pct is_stp_blocking stp_span_pct log_occupied_pct
------------ ------------ ------------------- ------------------ --------------- ------------ ---------------
      0           0                0                 45                 0              0            45

(return status = 0)
```

3. The query shows that there are no active transaction in the system. You can free all 45 percent of the occupied log space using the **dump transaction** command. After **dump transaction** is executed, the output of the same query shows:

```
has_act_tran OAtran_spid Act_log_portion_pct dump_tran_free_pct is_stp_blocking stp_span_pct log_occupied_pct
------------ ------------ ------------------- ------------------ --------------- ------------ ---------------
      0           0                0                 0                 0              0            0

(return status = 0)
```

## Viewing the Span of a Transaction

The system administrator can view the span of a transaction started by a specific process.

In this example, the transaction is identified by system process ID 15 and the database ID is 4:

```
select loginfo(4, 'xactspanbyspid', 15)
    as xact_span_spid15_dbid4
```

```
xact_span_spid15_dbid4
 ---------------------
                     10
```

This indicates that system process 15 is an active transaction and the log transaction span is 10 percent.

## Viewing the Oldest Active Transactions

The system administrator can view the processes that are using the most log space.

This example shows the top three oldest active transactions having longest spans in the transaction log:

```
select top 3 convert(numeric(3,0),
      loginfo(db_id(), ‘xactspanbyspid’, t.spid)) as XACTSPAN,
          convert(char(4), t.spid) as SPID,
          convert(char(20), t.starttime) as STARTTIME,
          convert(char(4), p.suid) as SUID,
          convert(char(15), p.program_name) as PROGNAME,
          convert(char(15), p.cmd) as COMMAND,
          convert(char(16), p.hostname) as HOSTNAME,
          convert(char(16), p.hostprocess) as HOSTPROCESS
from master..systransactions t, master..sysprocesses p
where t.spid = p.spid
order by XACTSPAN desc
```

```
XACTSPAN SPID     STARTTIME     SUID PROGRAM COMMMAND  HOSTNAME
HOSTPROCESS
-------- ---- ----------------- ---- ------- -------- ----------
-----------
  38      19  Aug 5 2013 12:20AM  1    ISQL   WAITFOR linuxstore4
26141
  20      20  Aug 5 2013 12:20AM  1    ISQL   WAITFOR linuxstore2
23467
  10      21  Aug 5 2013 12:21AM  1    ISQL   WAITFOR linuxstore6
4971
(return status =0)
```

# EMC PowerPath Device Fencing

Use the **SYBASE_MAX_MULTIPATHS** environment variable to override the number of paths iterated when PowerPath is enabled.

On the AIX platform, the method used for fencing EMC PowerPath devices is that the path of each device is fenced separately, which can result in performance issues during startup.

To increase startup performance when PowerPath is enabled, set the **SYBASE_MAX_MULTIPATHS** environment variable to 1, which will cause Adaptive Server to fence each device only once and allow PowerPath to broadcast the fencing operation to all of the paths.

- ksh: **export SYBASE_MAX_MULTIPATHS=1**
- csh: **setenv SYBASE_MAX_MULTIPATHS 1**

# Cyclic Redundancy Checks for dump database

Adaptive Server adds a cyclic redundancy check for accidental changes to raw data for database or transaction dumps created with compression to check and for verification that the compression blocks can be correctly read and decompressed.

The syntax is:

```
dump database database_name to dump_device with
compression=n,verify={crc | read_after_write}
load database database_name from dump_device with verify[only]=crc
```

Where:

- **verify=crc** – indicates that you are performing a cyclic redundancy check.
- **verify=read_after_write** – Backup Server rereads every compressed block after writing and decompresses it. If Backup Server finds an error, it prints the offset in the file it finds the error. **verify=read_after_write** is only applicable with the **dump database** command.

This example verifies database `new_dump` before dumping it to the `mydumpdev` device:

```
dump database new_dump to mydumpdev with
compression=x,verify=read_after_write
```

This example performs a cyclic redundancy check as it loads the `new_dump` database dump:

```
load database new_dump from mydumpdev with verify[only]=crc
```

If the database was not dumped with **verify=crc**, this option is ignored

This example performs a cyclic redundancy check and rereads every compressed block before dumping database `new_dump` to the `mydumpdev` device:

```
dump database new_dump to mydumpdev with
compression=x,verify=read_after_write, verify=crc
```

Usage:

- Dump created without the **verify=crc** parameter use the same format as earlier versions of Backup Server.
- You cannot load dumps that include cyclic redundancy checks with versions of Backup Server that do not include this functionality.
- **verify={crc | read_after_write}** checks are applicable only for files created using the **with compression** parameter.
- **verify=crc** works with any file type, including raw devices, disk files, tapes, pipes, or APIs. However, **verify=read_after_write** requires a ‘seek back’ for rereading the block, and is applicable only with raw devices and disk files.

- Adaptive Server ignores, and does not raise an error message, if you include **verify={crc | read_after_write}** parameters that are not applicable.

# Improvements to the Recovery Log Scan During load database and load tran

You can apply configuration changes for the default data cache to include a large buffer pool for reading log pages and asynchronous prefetch limits.

By default, boot time recovery automatically reconfigures the default data cache to include a large buffer pool for reading log pages and changes the asynchronous prefetch limits for both this pool and the default pool.

This is done to allow recovery to read log pages efficiently and to allow prefetch for large numbers of data (and index pages) and log pages. The settings applied by boot time recovery are:

- Large buffer pool (for example 128K for 16K page size) being 40% of the original page-sized pool size.
- The page size pool (for example 16K for a 16K page size) begin 60% of the original page-sized pool size.
- The **global async prefetch limit** for both pools set to 80% so that 80% of the pool can be used for pages that have been prefetched.

Prior to 15.7 SP60, these changes were not applied to **load database** and **load tran** recovery because these commands are potentially done on a live system and the changes could negatively impact the rest of the system.

As of 15.7 SP60, you can specify whether or not to apply these same configuration changes during **load database** and **load tran** recovery. To apply the configuration changes, use the **sp_configure** configuration parameter **enable large pool for load**. The settings for **enable large pool for load** are:

- 0 - changes are not applied during **load database** and **load tran** recovery.
- 1 - changes are applied during **load database** and **load tran** recovery. When making this change, be aware that load recovery might then impact other live (production) databases that are using the default data cache.

### See also

- *Configuration Parameters* on page 35

# Improvements in Prefetch Used by Recovery

The look-ahead size for prefetching pages of the recovery scan processes can be optimized using the **recovery prefetch size** configuration parameter.

Boot time, **load database**, and **load tran** recovery can prefetch to-be-recovered data pages from disk into cache so that recovery does not need to wait when attempting to redo or undo log records.

Prefetching pages is done by having an auxiliary process operate ahead of the recovery redo or undo scan reading data (and index) pages from disk into cache. Problems can occur if the prefetch auxiliary process:

- Operates too far ahead of the recovery scan – in this case, the recovery scan might find that the pages are no longer in cache, as they have been aged out.
- Operates too close to the recovery scan – in this case, the recovery scan might block the waiting of prefetch disk reads to complete.

In 15.7 SP60, the look-ahead size can be optimized using the **recovery prefetch size** configuration parameter. The look-ahead size can be set to be dynamically determined, or you can specify the exact size of the look-ahead in numbers of log records using the **recovery prefetch size** configuration size parameter.

The settings for **recovery prefetch size** are:

- 0 - dynamic look-ahead is applied. That is, SAP ASE determines the optimal look-ahead size. This is the default.
- value other than 0 - the value specifies the exact size of the look-ahead in numbers of log records. Use this only if you find a particular look-ahead size works more effectively that dynamic look-ahead.

### See also

- *Configuration Parameters* on page 35

# sp_monitorconfig Permission Changes

The permission requirements for **sp_monitorconfig** have changed.

To execute **sp_monitorconfig**:

- With granular permissions enabled, you must be a user with mon_role or have `manage server` privilege.
- With granular permissions disabled, you must be a user with either mon_role or sa_role.

# Modify the ELC Size

The configuration option **engine local cache percent** configures Engine Local Cache (ELC) size for procedure cache memory.

Based on this configuration option, SAP ASE will configure procedure cache ELC size as given percentage of **procedure cache size**.

- Default value for this configuration is 50. Which means ELC size will be 50% of **procedure cache size**.
- It is a static option; SAP ASE needs to be rebooted for the option to take effect.
- This configurations option can be used only when large ELC is enabled (using boot time traceflag 758).
- For optimal performance, a value no larger than 80 is recommended.

Example:

```
1> sp_configure "engine local cache percent",70
2> go
Parameter Name Default Memory Used Config Value Run Value Unit Type
-------------- ------- ----------- ------------ --------- ---- ----
    engine   local cache percent  50   0   70  50  percent static
(1 row affected)

Configuration option changed. Since the option is static,
Adaptive Server must be rebooted in order for the change to take
effect.
Changing the value of 'engine local cache percent' does not
increase the amount of memory Adaptive Server
```

### See also

- *Configuration Parameters* on page 35

# TSM Client Multithread Support

Backup Server provides multithread support needed for Tivoli Storage Manager (TSM) options such as LAN-FREE.

To enabled multithread support for TSM options, use the –D8192 option during Backup Server startup.

For example:

```
${BACKUPSERVER} -SSYB_BACKUP -I$SYBASE/interfaces -M${SYBMULTBUF} -
D8192&
```

This feature is not enabled by default.

# Abstract Plan Sharing Between Different Users

Enable abstract plan sharing between different users by using either the session wide option **set plan shared**, or the server wide option **abstract plan sharing**.

In previous releases, a separate entry was needed for each user ID executing a query using abstract plans.

The session wide syntax is:

```
set plan shared {on | off}
```

The server wide syntax is:

```
sp_configure "abstract plan sharing", 1
```

**Note:** Tables of a query using a shared abstract plan must be explicitly prefixed with the owner name or belong to the DBO user.

### See also

- *Configuration Parameters* on page 35

# System Changes

System changes for 15.7 SP60.

## Configuration Parameters

New and changed configuration parameters for 15.7 SP60.

*New Configuration Parameters*

**Table 1. abstract plan sharing**

| | |
|---|---|
| Default value | 0 (off) |
| Range of values | 0 (off), 1 (on) |
| Status | Dynamic |
| Display level | Comprehensive |
| Required role | System Administrator |
| Configuration group | SQL Server Administration |

Enables abstract plan sharing between different users. Tables of a query using a shared abstract plan must be explicitly prefixed with the owner name or belong to the DBO user.

See *Abstract Plan Sharing Between Different Users* on page 33

**Table 2. enable large pool for load**

| | |
|---|---|
| Default value | Dependent on **enable functionality group**. |
| Range of values | 0 (off), 1 (on) |
| Status | Dynamic |
| Display level | Comprehensive |
| Required role | System Administrator |
| Configuration group | SQL Server Administration |

Configure the use of large buffer pools during the recovery phase for **load database** and **load transaction** commands.

The **enable large pool for load** configuration parameter defaults to 0 if **enable functionality group** is set to 0, and to 1 if it is set to 1.

See *Improvements to the Recovery Log Scan During load database and load tran* on page 23

**Table 3. engine local cache percent**

| | |
|---|---|
| Default value | 50 |
| Range of values | 0 - 100 |
| Status | Static |
| Display level | Comprehensive |
| Required role | System Administrator |
| Configuration group | SQL Server Administration |

Configure Engine Local Cache (ELC) size for procedure cache memory.

See *Modify the ELC Size* on page 29

**Table 4. recovery prefetch size**

| | |
|---|---|
| Default value | 0 (use dynamic prefetch) |
| Range of values | 0 - 20,000 |
| Status | Dynamic |
| Display level | Comprehensive |
| Required role | System Administrator |
| Configuration group | SQL Server Administration |

Sets the look-ahead size (in numbers of log records) to be used by the recovery prefetch scan. Set to 0 if the scan is to determine the look-ahead size dynamically, or to a value > 0 if the look-ahead size is to be set to a specific number of log records to look-ahead.

See *Improvements in Prefetch Used by Recovery* on page 25

### *Changed Configuration Parameters*

The configuration parameter, **enable plan sharing** is no longer dependent on the **enable functionality group** parameter.

**Table 5. enable plan sharing**

| | |
|---|---|
| Default value | 0 (off) |
| Range of values | 0 (off), 1 (on) |
| Status | Dynamic |

| | |
|---|---|
| Display level | Comprehensive |
| Required role | System Administrator |
| Configuration group | Query Tuning |

## Functions and Procedures

New and changed functions and procedures for 15.7 SP60.

| Function | |
|---|---|
| **loginfo** – extended to further support managing transaction log space. | *loginfo* on page 5 |

| Parameter | |
|---|---|
| **sp_xact_loginfo** – provides the span of oldest active transaction in terms of percentage of total log space. | *sp_xact_loginfo* on page 8 |
| **sp_monitorconfig** – permission requirements for sp_monitorconfig have changed. | *sp_monitorconfig Permission Changes* on page 27 |

# Index