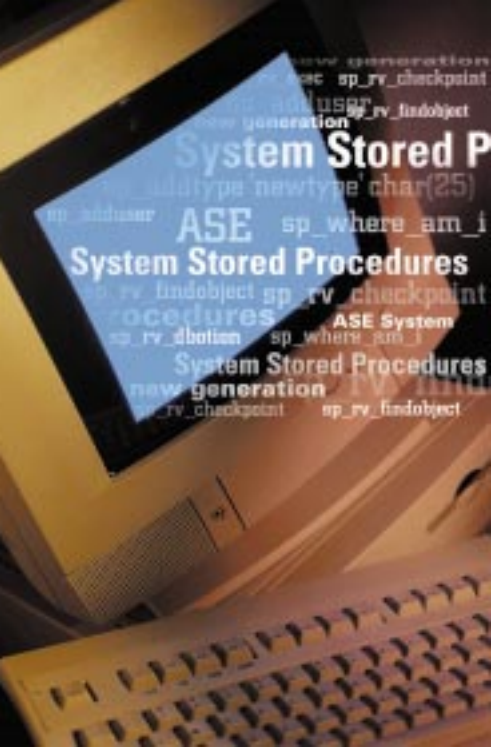

A New Generation of ASE System Stored Procedures

By Rob Verschoor

Creating a new generation of ASE System Stored Procedures to give DBAs more control in multi-database environments

Rob Verschoor is a freelance consultant in The Netherlands, specializing in Sybase performance and tuning issues. He can be reached at rob@sypron.nl.

ASE System Stored Procedures, such as "sp_help" and "sp_who", are frequently used by DBAs and ASE users to perform a wide variety of tasks. The idea behind System Stored Procedures (SSPs) is that they can be executed from any database in the server, while only a single copy of the SSP exists in the "sybsystemprocs" database.

While this principle works very well, the functionality of the existing traditional SSPs is somewhat limited because they can only access databases with names that are specified at compile time, as well as the executing session's current database. Many DBA tasks require a more flexible functionality.

This article presents a method for constructing a new generation of truly generic, platform-independent SSPs which are more flexible and powerful than traditional SSPs. Based on two little-known features of Transact-SQL, these new SSPs can access any number of different databases during a single execution, which allows very powerful functionality to be implemented that would otherwise not be feasible.

Why Would You Need New SSPs?

In practice, DBA tasks often cover multiple databases in the same server. For example, existing relations between data elements in different databases imply that these databases should not be treated as isolated entities. Also, common DBA issues such as database maintenance activities, administrating database access rights, etc., often span database boundaries, as the examples in this article will illustrate.

Working in different databases traditionally requires that the T-SQL **use database** command be issued to change from one database to another. However, because the **use** command cannot be part of a stored procedure (nor of any other compiled object), DBAs often write their own tools which are external to the server (such as Unix shell scripts), to perform such tasks more efficiently. These effectively generate the required T-SQL statements—including the necessary **use** commands—and then execute these statements.

The disadvantage of such operating system-level tools is that they require extra development in a non-database language, increasing the overall complexity and cost of the DBA environment. As these disadvantages do not stop DBAs from creating such external tools, it seems that this class of practical DBA requirements is not well addressed by existing SSPs.

In contrast, the new SSPs presented in this article are capable of performing operations in any number of different databases in a single execution, giving DBAs more control over multi-database environments and allowing them to manage cross-database issues in a more effective manner. In many cases, these SSPs will also remove the need for using OS-level tools, making DBAs more productive.

Note that the new SSPs require System 10 or later, meaning that practically all Sybase users can benefit from this functionality.

Example: Setting Database Options More Easily
As a simple example, let's look at the well-known SSP "sp_dboption," which is used to set database options (such as "select into/bulkcopy," "read only," etc.). The DBA must enter a series of commands which looks something like this:

```
1> use master
2> go
1> exec sp_dboption my_db, "read only", true
2> go
Database option 'read only' turned ON for database 'my_db'.
Run the CHECKPOINT command in the database that was changed.
(return status = 0)
1> use my_db
2> go
1> checkpoint
2> go
```

Although most Sybase DBAs are familiar with this somewhat clumsy command sequence, the task of setting database options can be greatly simplified by using the new SSP "sp_rv_dboption," which conveniently replaces the entire command sequence above by just a single SSP call:

```
1> exec sp_rv_dboption my_db, "read only", true
2> go
Database option 'read only' turned ON for database 'my_db'.
Run the CHECKPOINT command in the database that was changed.
...CHECKPOINT performed in 'my_db'.
(return status = 0)
```

Note what happens here: The SSP sp_rv_dboption calls the traditional SSP sp_dboption, but also automatically performs the checkpoint in the affected database. Furthermore, sp_rv_dboption can be invoked from any database, not just from *master*, as sp_dboption requires. This means it is not necessary to run separate **use** commands anymore.

This is a major difference with the traditional sp_dboption. Not only there is less typing to do when setting a database option using sp_rv_dboption, but because the need for use commands has been eliminated, database options can now also be modified from stored procedures, which can call sp_rv_dboption.

So how does this work? Basically, the trick of sp_rv_dboption lies in two little-known features of T-SQL.

Feature #1: Controlling The Database Context for an SSP
It is commonly known that SSPs execute within the context of the database from which they are invoked, even though the SSPs themselves reside in a different database (sybsystemprocs). However, the SSP mechanism provides some additional functionality which is not widely known at all. This functionality is illustrated by the following simple SSP "sp_where_am_i", which displays the name of the current database:

```
1> use sybsystemprocs
2> go
1> create procedure sp_where_am_i
2> as select db_name()
3> go
1> use my_db
2> go
1> exec sp_where_am_i
2> go
------------------------------
my_db

1> exec other_db..sp_where_am_i
2> go
------------------------------
other_db
```

The first call to sp_where_am_i displays "my_db" as expected, because the SSP is executed from the database my_db. This is the normal behaviour as we all know it; in fact, this is how sp_help always shows a list of objects in the current database.

The second call shows the trick: though the current database is still my_db, "sp_where_am_i" surprisingly displays the name "other_db". This is because the SSP is executed in the context of a different database (other_db) by prefixing the procedure name with the name of that database.

Traditionally, one would normally first change the database context with the command **use other_db**, and then issue **exec sp_where_am_i**. By calling the SSP in the above manner, the **use** command can be avoided while the same effect is achieved.

There are many practical applications of this feature. For example, to display the layout of a table in another database, it is not necessary to change databases anymore, because it is sufficient to simply run the command **exec other_db..sp_help some_table**. (Note: As always, the keyword "exec" can be

omitted when this is the first command in the batch.) By avoiding the **use** command this way, interactive T-SQL sessions are more efficient, increasing productivity. It is remarkable that this feature is so little-known while it has existed since version 4.x (and probably earlier), which is probably due to the fact that has not been documented explicitly until ASE 11.5.

Feature #2: Executing a Variable Stored Procedure

Sybase System 10 introduced the possibility of executing a stored procedure by specifying its name through a variable:

```
1> declare @proc varchar(100)
2> select @proc = "sp_help"
3> exec @proc
4> go

1> declare @proc varchar(100)
2> select @proc = "other_db..sp_help"
3> exec @proc some_table
4> go
```

This can be seen as a limited way of using "dynamic" SQL, which is nonetheless quite useful. While this feature is not formally documented, many Sybase users rely on this functionality, as do some of Sybase's own SSPs. For these reasons, this feature is not likely to disappear from future ASE releases.

Putting It Together: sp_rv_dboption

By combining the two T-SQL features shown above, sp_rv_dboption can be constructed. First, the SSP "sp_rv_checkpoint" is created. This SSP simply performs a checkpoint, but by applying feature #1 above ("controlling the database context"), any database can be checkpointed:

```
1> use sybsystemprocs
2> go
1> create procedure sp_rv_checkpoint
2> as checkpoint
3> go
```

The SSP sp_rv_dboption can now be created as follows (note that this is a simplified version for clarity) :

```
1> create procedure sp_rv_dboption
2>  @db varchar(32),
3>  @opt varchar(32),
4>  @val varchar(32)
5> as
6>  declare @proc varchar(100)
7>  exec master..sp_dboption @db, @opt, @val
8>  select @proc = @db + "..sp_rv_checkpoint"
9>  exec @proc
10>  print "...CHECKPOINT performed in '%1!'", @db
11> go
```

As can be seen from this code, the two T-SQL features discussed above give sp_rv_dboption its special functionality. First, on line 7, sp_dboption is executed within the context of the master database by applying Feature #1, so there is no need anymore to explicitly change to the master database with the **use** command in order to run sp_dboption. Second, on lines 8/9, the database whose option was set is checkpointed automatically by using "sp_rv_checkpoint", combining both features.

While sp_rv_dboption is a rather simple SSP, it is a good illustration of how the underlying T-SQL techniques can be applied. We will now look at more advanced SSPs based on these same two T-SQL features. All SSPs discussed in this article will work in any ASE server running version 10 or later, and are available for download (see end of this article for the web address).

Execute in All Databases: sp_rv_exec

DBAs often need to perform identical actions in different databases. For example, to give the new employee "jsmith" access to five databases, the command **sp_adduser 'jsmith'** must be executed five times, once in every database. Another example is adding a new user-defined datatype: this requires running a command such as **sp_addtype 'newtype', 'char(25)'** in all databases where this datatype is needed.

Traditionally, one has to run the required command as many times as needed, issuing **use** commands in between to switch databases. When managing a server with a large number of databases, this can be a rather boring job, for which DBAs often write shell script-like tools to make their lives a bit easier.

However, this type of problem can now finally be dealt with in a much more elegant way, by means of the SSP "sp_rv_exec," which lets you execute your SSP in all databases in the server with just a single command, no matter how many databases there are:

```
exec sp_rv_exec "sp_adduser", "jsmith"
-- adds user "jsmith" to all databases
exec sp_rv_exec "sp_addtype", "newtype", "char(25)"
-- adds datatype "newtype" to all databases
```

sp_rv_exec is based on the two T-SQL features discussed above. Basically, it consists of a loop iterating over the databases which exist in the server. For each database, the SSP specified as the first parameter is then executed in the context of that database, along with the other parameters specified.

It may not be desirable to add users or datatypes to certain databases, such as *master* or *tempdb*. While the default behaviour is to perform the specified action in all databases in the server, the scope of action can be limited to a specific set of databases as required. The downloadable files mentioned above contain documentation on how to define this.

A Swiss Army Knife for DBAs: sp_rv_findobject

By building additional functionality on top of sp_rv_exec, some very powerful SSPs have been created; probably the most versatile of these is "sp_rv_findobject". This SSP basically searches some or all databases in the server for objects matching a number of search criteria. Here are some examples:

```
(1) sp_rv_findobject "name=[Aa]%"

(2) sp_rv_findobject "type=U", "owner!=dbo"

(3) sp_rv_findobject "type=V", "created>=01-May-99"

(4) sp_rv_findobject "coltype=%identity%"

(5) sp_rv_findobject "coltype=%identity%", "display=DB.OW.NM"

(6) sp_rv_findobject "type=UDD", "output=abc"
```

The first example will print a list of all objects in the server (i.e. tables, views, rules, triggers, etc.) having a name starting with "A" or "a", regardless of the database in which the object resides. The second example lists all user tables which are not owned by the database owner. The third will find all views created after April 30, 1999. As these examples illustrate, different search criteria can be specified to form a search filter; they combine as a logical "AND".

The fourth example will list the fully qualified column names (e.g., dbname.owner.tablename.colname) of all identity columns in the server. But suppose we only want a list of the tables which contain an identity column, and not the column names themselves. In this case, the "display=" modifier can be used, as in the fifth example. This specifies which information should be displayed in the final result; only the fully qualified table name (e.g., dbname.owner.tablename) is printed; the column name is suppressed.

By default, the search results are displayed on the client screen. Instead, by using the "output=" option as in the last example, the search results are written to a table in tempdb; in this case, the results table will be named "tempdb..abc".

Note that there are more search options than can be discussed here; a full overview is included in the downloadable files.

While this server-wide search capability is useful in itself, the most powerful feature of this SSP is the option to perform operations on the individual objects found. For example, let's assume we want to display the existing indexes on all tables which have the word "sales" in their names. The first example below will find all these tables, and the execute "sp_helpindex" on each of them:

```
(1) sp_rv_findobject "name=%sales%", "type=U",
    "exec=sp_helpindex"

(2) sp_rv_findobject "owner=jsmith", "exec=sp_rv_dropobject"
```

In the second example, we want to drop all objects owned by user "jsmith"; this person has left the company and did not clean up the many tables, views, etc., he left hanging around the various databases. By running the command shown here, all objects owned by "jsmith" are found. Then, the SSP "sp_rv_dropobject" is executed for each of these objects. sp_rv_dropobject is a fairly powerful (or dangerous, depending on your viewpoint) SSP, which drops any type of object whose name is passed as a parameter.

There is hardly a limit as to what is possible: A DBA can write his own SSPs to perform custom tasks and run these against objects around the server through "sp_rv_findobject"; in fact, sp_rv_dropobject is an example of this approach.

Statistics Made Easy: sp_rv_update_statistics

A common DBA task is to ensure that the distribution statistics of all indexes be regularly updated using the **update statistics** command. Typically, a batch job is created for this purpose, which runs every now and then to perform "update statistics" and "sp_recompile" for all user tables in all databases (note that "sp_recompile" is often forgotten!). Because new user tables may be created, this batch job is usually a multi-step process which first generates the necessary "update statistics" and "sp_recompile" statements for all user tables in every database, and then executes these statements.

Because this job usually covers different databases, **use** commands are traditionally required to change the database context during this process. Many DBAs have written shell script-based tools to automate this entire process, so that it will work correctly regardless of the number of existing databases or tables.

The SSP "sp_rv_update_statistics", which is based on sp_rv_exec, can replace this entire batch job by just a single command. In its simplest form (the first example below), it will run "update statistics" and "sp_recompile" on all user tables in all databases without the need for any other tools:

```
(1) sp_rv_update_statistics "execute"

(2) sp_rv_update_statistics "generate"
```

This first example requires ASE version 11.5 or later, because it uses CIS features to actually perform the **update statistics** command.

When running an earlier version than 11.5, the second example (with the "generate" parameter) should be used. In this case, sp_rv_update_statistics will generate the complete T-SQL script for updating the statistics, which should then be executed with isql. While the first example is clearly most convenient, the second example is still a much more efficient way to generate the required T-SQL statements than the traditional approach which uses OS-level tools, external to the server.

Who's Who in the Server: sp_rv_helplogin

Suppose a DBA needs to remove the login "jbrown" from the server, because this person has left the company. The DBA executes the command **sp_droplogin jbrown**, but gets the error message "User exists or is an alias or is a database owner in at least one database." Unfortunately, this message doesn't indicate in which of the 35 or so databases in the server this login still owns something, so the DBA has no choice but to check each database individually until the **sp_droplogin** command succeeds. Needless to say, this isn't the most fascinating of tasks.

For this type of problem, "sp_rv_helplogin" is probably useful. This SSP, also built on top of sp_rv_exec, displays how a specific login will access each database in the server, i.e., to which database user this login corresponds in every database. Furthermore, it shows whether this login still owns any objects in every database. From this output, it becomes clear immediately where "jbrown" still had some objects, datatypes or aliases left, so the DBA doesn't have to waste time on working this out.

To display the database users for "jbrown," the following command should be used:

```
exec sp_rv_helplogin "jbrown"
```

The main virtue of "sp_rv_helplogin" is that, unlike traditional SSPs, it shows a complete picture of how (i.e., as which database user) a login will access every database.

Download and Try It Yourself

All SSPs mentioned in this article are available as copyrighted freeware, and can be downloaded from http://www.euronet.nl/~syp_rob/download.html.

Due to space limitations, it is not possible to discuss all aspects of these new SSPs in this article. The downloadable file contains a detailed reference on the command options and possible applications.

Relation to ASE 12.0: Execute Immediate

The next major release of ASE (version 12.0, a.k.a. Avatar) will contain the long-awaited "execute immediate" feature. This makes it possible to execute a T-SQL command string which is dynamically created in a character variable. As many ASE users expect this feature to deliver significant new flexibility, one might wonder whether the existence of the new-generation SSPs described in this article is still justified.

It certainly is. The "execute immediate" functionality only overlaps with Feature #2 mentioned earlier (executing a variable stored procedure); Feature #1 (controlling the database context), being the other building block of these SSPs, is not related to "execute immediate" at all. Furthermore, "execute immediate" does not support the **use** command, while the new SSPs will also work in ASE versions 10.x to 11.9.x. ■