
Title: SQL Server error trapping

Platform: Microsoft SQL Server

Creation date: 2002

Author: Keith Biddle (MCSE)
Verstand Ltd.
www.SqlserverPortal.com

1 Table of contents

1 Table of contents.....	2
2 Overview	4
2.1 Subject	4
2.2 Summary	4
2.3 Audience	4
2.4 Limitations	4
2.5 Acronyms	4
2.6 Glossary	4
3 Introduction	5
4 Why use SQL error trapping?	5
5 Requirements for SQL error trapping.....	5
6 Types of errors	6
6.1 Execution time errors	6
6.2 Business logic errors.....	6
6.3 Domain / attribute integrity	6
7 SQL generated errors	7
7.1 SQL error codes.....	7
7.2 Return codes	7
8 User generated error codes	8
8.1 SQL error codes.....	8
8.2 Return codes	8
8.3 OUTPUT parameter	8
8.4 Error Resultset	8
9 Using Return()	9
9.1 SQL defined return codes	9
9.2 defined return codes	9
10 Using OUTPUT.....	10
11 Using transactions & rollback.....	11
12 Error handling with TXN code.....	12
13 Logging the errors	13
13.1 In the App.....	13

13.2	In SQL.....	13
14	Logging errors to a table	14
14.1	usp_DBA_Template_LogErrors.....	15
14.2	usp_DBA_LogError.....	16
14.3	Objects created locally in each DB.....	20
15	Generating test errors.....	21
15.1	FK error.....	21
15.2	Identity insert error.....	21
16	Code that receives the error info.	22
16.1	Javascript - ASP	22
16.2	Vbscript – ASP.....	23
16.3	TSQL.....	24
16.4	C++	24
17	Retrieving the error info	24
18	References.....	24

2 Overview

2.1 Subject

This document establishes the programming standards for SQL server development . Dealing with SQL error handling & its integration into C++, ASP & TSQL.

2.2 Summary

The document establishes 'best practice' on SQL server error trapping.

2.3 Audience

All developers & DBA's should read this document.

2.4 Limitations

These standards apply to development on Microsoft SQL Server. Some of the SQL examples shown may only work on SQL Server 7. MS SQL Server 7.0 is mostly compliant with ANSI SQL-92.

2.5 Acronyms

Acronym or Abbrev.	Desc
app	Application
ANSI	American National Stds. Inst.
BOL	Books On Line (SQL)
DDL	Data definition language eg create table
DML	Data manipulation language eg select
DRI	declarative referential integrity
DB	database
Err	error
RI	referential integrity
Sp	Stored procedure
Tbl	table
Txn	transaction
TSQL	Transact SQL
Usp	User stored proc

2.6 Glossary

Term	Desc
Stored procedure	A block of compiled TSQL, stored as an object in MS SQL Server
Error handling	Catching & processing of errors
Error trapping	Catching & processing of errors

3 Introduction

Error trapping in SQL Server, although not as good as a full programming language, is still implementable. This document outlines the various ways to implement SQL error trapping, and recommends a number of methods. It then describes a possible standard that ties in with apps error trapping & logging.

4 Why use SQL error trapping?

- It is an important part of 'good practice' SQL programming.
- It improves reliability.
 - It can control flow after non-fatal errors.
 - It provides feedback to the calling app, when an error occurs in an sp.
- It improves data integrity.
 - It can rollback a transaction to prevent partial commits.
- It helps troubleshooting for dev & ops.
 - Logging needs error info, sql error trapping can provide this.
 - It can maximise the information logged & returned to the app.
- It's just a SELECT statement, what can go wrong?
 - You can get arithmetic overflow due to calculations on a select.
 - (Ref: SQL pro mag, feb 2002, Nilsson.)

5 Requirements for SQL error trapping

Essential:

- Be the same for both TXN & non TXN code.
- Trap for non-fatal errors, SQL error number & rowcount.
- Minimise code in TSQL & C++/ASP.
- Return success / fail to calling procedure (usp or app)
- Be able to work with C++, jscript server side, TSQL, VB.

Desirable:

- Provide debug print on/off functionality
- Return SQL error number to calling procedure (usp or app)
- Provide some form of catching method for fatal errors
- Provide step / crash point info, ie where error occurred in usp.
- Return user defined error codes for business log tests

Data to be logged:

- Application
- DB name
- Stored Procedure
- Step (location of failure)
- SQL error number
- SQL error msg (lookup to sysmessages)
- Business logic error number
- Business logic error msg
- Rowcount

6 Types of errors

There are several types of errors that we need to plan for:

1. Execution time errors due to: code bugs, unexpected data, etc. Fatal & non-fatal.
2. Business logic errors
3. Domain or attribute integrity errors ie value range

6.1 Execution time errors

Non-fatal errors will allow the usp to continue, even though a call has failed, eg 0 rows rtrnd or foreign key failures

Fatal errors will crash the sp.
eg if a table is missing.

This will NOT send a value back in return() statement. So we must trap for this.

6.2 Business logic errors

They relate to business logic rules. They are difficult to specifically trap for, because the code might work, but the result might not be what you wanted. However, you can reduce the risk of business logic errors by using domain integrity methods.

If a usp fails a business logic test, then you may wish to return error information about this, such as 'no records found'. This can be done via:

- user defined SQL errors, using RAISERROR
- custom RETURN values
- OUTPUT params.

6.3 Domain / attribute integrity

These are the measures we take to restrict the type of values that can be entered into a column. We use the following to achieve this:

- Defaults
- Constraints
 - foreign key
 - check
 - Not null
- Unique indexes
- Data types
- User defined data types
- SQL error trapping
 - IF..ELSE, CASE, etc

7 SQL generated errors

These occur in 2 forms:

7.1 SQL error codes

An error occurs, & the SQL error code is placed into @@error global variable. This is cleared with the next execution, so must be tested for immediately after each SQL statement.

7.2 Return codes

An error occurs, the default return code of 0 (success), is changed to a negative integer (-1 to -99). To a certain extent, the SQL error code is more informative.

8 User generated error codes

For situations where a business logic error occurs, we may want to return a code to say it has failed the business logic, even though it has not caused a SQL error or return error.

8.1 SQL error codes

RAISERROR can generate user defined SQL error codes.
50,001 & above ok for user defined error messages.

BOL =

"trace flag 2701 Sets the @@ERROR system function to 50000 for RAISERROR messages with severity levels of 10 or less."

"Acceptable values for user-defined error messages start with 50001."

When an error is raised, the error number is placed in the @@ERROR function, which stores the most recently generated error number. @@ERROR is set to 0 by default for messages with a severity of 1 through 10.

To use RAISERROR, you need to add an error msg to master..sysmessages using sp_addmessage. It can then be called by RAISERROR, to generate a user defined SQL error.

Risk is that may lose data on master..sysmessages if move server or restore DBs. Also it appears the severity flag in the error msg entry, will effect its actions, eg

@@ERROR is set to 0 by default for messages with a severity of 1 through 10.

If severity 20-25 then it is fatal, & will crash the usp.

This could really confuse your error trapping & affect reliability.

Don't use it.

8.2 Return codes

You can use RETURN to specify your own values; just return a positive integer, & interpret this in your calling procedure. Or a return value table could be created to hold user defined return code descriptions.

8.3 OUTPUT parameter

Use the stored procedure OUTPUT parameter to return a value, & interpret this in your calling procedure. Or a return value table could be created to hold user defined return code descriptions. Use BusinessLogicMessages tbl in each local DB for this.

8.4 Error Resultset

Pass error or business logic values back to calling proc, via a resultset.

As a 2nd rst or as an extra field in a rst?

Works in Asp but a bit tricky.

I suggest we don't use this because it is not generic across languages.

9 Using Return()

9.1 SQL defined return codes

Normally, completed stored procedures return a status number of 0 ie success.
If a RETURN statement is not used and an error occurs when the stored procedure is executed, SQL Server can return one of the following values:

Value	Description
-1	Missing object
-2	Data type error
-3	Process was chosen as deadlock victim
-4	Permission error
-5	Syntax error
-6	Miscellaneous user error
-7	Resource error, such as out of space
-8	Nonfatal internal problem
-9	System limit was reached
-10	Fatal internal inconsistency
-11	Fatal internal inconsistency
-12	Table or index is corrupt
-13	Database is corrupt
-14	Hardware error

The values -15 through -99 are reserved for future SQL Server expansion.

Simplest way to trap for failures is to rely on the SQL generated rtn codes:

Success = 0

Failures = -1 to -99

So can assume failures are values NOT EQUAL to 0, or MORE THAN 0.

9.2 User defined return codes

A RETURN statement in a stored procedure can be used for business logic return status numbers.

If you don't want to use SQL error logging. You can use 'Return codes' to pass back values to the calling usp or app. There are two ways to do this:

1. Pass back an integer, & interpret this in your code.
2. Pass back integers >50,001, thus clearly not 'SQL svr error ID's'. Enter a row in BusinessLogicMessages tbl, then retrieve the error msg from this tbl.

10 Using OUTPUT

If want to rtn more than one integer value (ie using Return codes), then use OUTPUT parameters. These can give values back to the calling routine eg err num, err msg, step. The following example shows how to use OUTPUT parameters.

```
USE pubs
```

```
go
```

```
create table table1 (col1 varchar(10))
```

```
go
```

```
create proc usp_testrtnmsgs @piErrNum int OUT AS
```

```
insert into table1 values('xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx') --this generates a SQL error
```

```
select @piErrNum = @@error
```

```
if @piErrNum > 0 goto ERR_HANDLER
```

```
return(0) --Success
```

```
ERR_HANDLER:
```

```
    return(1) --Failure
```

```
go
```

```
declare @iErrnum int
```

```
declare @iRtnCode int
```

```
execute @iRtnCode = usp_testrtnmsgs @piErrnum = @iErrnum OUTPUT
```

```
print 'SQL error num = ' + @iErrnum
```

```
print @iRtnCode
```

```
drop table table1
```

```
go
```

```
drop proc usp_testrtnmsgs
```

```
go
```

11 Using transactions & rollback

A transaction usually contains several statements to fulfill some business logic. Using transaction code can avoid partially committed transactions. Tables should be accessed in a predefined order eg a-z to reduce the risk of deadlocking.

It should start with the statement 'begin transaction' and ends with a 'commit' if no error has occurred and 'rollback' if an error has been found. The careful implementation of the simple error checks will ensure that the transaction is committed correctly.

You should test them carefully to ensure deadlocking doesn't occur, use the A-Z table access plan.

12 Error handling with TXN code

This uses the return code, to pass back the SQL error number or business logic number.

0 if successful & (0-50,000) SQL or 50,001+ business logic.

If no return value then fatal SQL error.

Lookup SQL code in master..sysmessages

Lookup business logic code in a localDB..businesslogicmessages.

```
drop proc usp_DBA_TestErrTrapEG
go
-----
create proc usp_DBA_TestErrTrapEG as

declare @iStep int          --Range 5 to 1000+ (step 5, so can fill in)
declare @iErrNum int        --Range 0 to (-1 to -99)
declare @iRowCount int     --Range 0 to thousands
declare @iDebug int        --Range 0 or 1

select @iDebug = 1          --1 show, 0 not show.
select @iErrNum = @@error  --Should be 0
select @iStep = '0'
-----

BEGIN TRANSACTION

IF @iErrNum > 0 GOTO ERR_HANDLER
BEGIN
    select * from titleauthor where au_id = '172-32-1176'      --If no rows then fail
    select @iStep = 5, @iErrNum = @@error, @iRowCount= @@rowcount
    if @iDebug=1 print @iStep print @iErrNum print @iRowCount
    if @iRowCount = 0
    BEGIN
        --Assign business logic error code
        select @iErrNum = 50001 --If wanted to continue processing set @iErrNum = 0
        goto ERR_HANDLER
    END
END

END

-----
IF @iErrNum > 0 GOTO ERR_HANDLER
BEGIN
    insert into titleauthor (au_id,title_id) values('172-32-1176','PS3333')  --Fails
    select @iStep = 10, @iErrNum = @@error, @iRowCount= @@rowcount
    if @iDebug=1 print @iStep print @iErrNum print @iRowCount
END

-----
IF @iErrNum > 0 GOTO ERR_HANDLER
BEGIN
    COMMIT
    --Rtn 0 if no errors
    RETURN @iErrNum
END

-----
ERR_HANDLER:
    --Add any other cleanup code here.....
    ROLLBACK
    IF @iDebug=1 print 'ERR_HANDLER - ROLLBACK' print @iErrNum print @iRowCount
    RETURN @iErrNum
-----

--Running sp
declare @irtncode integer

select @irtncode = null      --Reset rtn code.
execute @irtncode= usp_TestErrTrapEG
select @irtncode
```

13 Logging the errors

Here are the options for logging error info.

13.1 In the App

Pass error info to the app to:

- display as a window
- log to a db
- local log file

The SQL error logging will be able to pass error info back to the app, which can then be logged to the local file.

13.2 In SQL

Could log errors to the event log, using xp_logevent (), but this will be cleared out with time & may fill it up.

Log errors to a sql table (no problem as we are already connected & in the db)

See the following EG's

14 Logging errors to a table

Returning values to a calling app, that aren't part of the resultset, means only one or two values can be returned. The answer to this problem is to log as much info as possible to a SQL error log table, & return an errorlogID, which the apps can use to retrieve the row.

An extra requirement is for the app to pass its App ID to the usp for logging. See platforminfo..applications (platforminfo is a db holding generic platform wide info such as information about the applications that are running, these apps are ones you have coded in C++, VB, etc.)

I have chosen to locate the SQLErrorLog & BusinessLogicMessages tables in each local DB, rather than platformInfo for a number of reasons:

- Speed
- Reliability
- Keeping the DB procedures atomic & within one DB.
- Crash resistant: ie not dependent on other servers.

The monitoring of the SQLErrorLog tbl will be via the intranet.

Each DB will need:

- usp_DBA_LogError
- SQLErrorLog table
- BusinessLogicMessages table
- usp_DBA_Template_LogErrors

The application must have a value entered into the db platforminfo..applications tbl.

Create your own business logic messages / rtn codes in businesslogicmessages tbl.

Create your usp, using usp_DBA_template_logerrors as a guide.

Pass the errorlogID back to your app, & it can retrieve the full info using

```
usp_DBA_GetSQLerrInfo @piAppID int, @piErrLogID
```

This can either be displayed or put into the std log file.

14.1 usp_DBA_Template_LogErrors

The following code demonstrates error trapping & logging to a sql table for both sql errors & business logic errors. **I recommend this method as a std for error handling & logging.** Add comment header & debug print code (see Intranet egs).

Create procedure **usp_DBA_Template_LogErrors** @piAppID int as

declare @iRowCount int, @iErrNum int, @iUspID int, @iStep int, @iErrLogID int

```
select @iUspID = (select @@procid)    --Rtns stored procedure ID
select @iErrNum = @@error            --Should be 0
select @iStep = '0'
```

```
-----
IF @iErrNum > 0 GOTO ERR_HANDLER
BEGIN
    insert into pubs.dbo.titleauthor (au_id,title_id) values('172-32-1176','PS3333')    --Fails
    select @iStep = 5, @iErrNum = @@error, @iRowCount= @@rowcount
END
```

```
-----
IF @iErrNum > 0 GOTO ERR_HANDLER
BEGIN
    select * from pubs.dbo.titleauthor where au_id = '172-32-1176' --If no rows then fail
    select @iStep = 10, @iErrNum = @@error, @iRowCount= @@rowcount
    if @iRowCount = 0
    BEGIN
        --Assign business logic error code
        select @iErrNum = 50001 --If wanted to continue processing set @iErrNum = 0
        goto ERR_HANDLER
    END
END
```

```
-----
/*
Rtn values:
0      success
1 up   SQL error log ID (identity value retrieved from SQLerrorlog)
-1     a failure to log
*/
```

```
IF @iErrNum > 0 GOTO ERR_HANDLER
BEGIN
    return 0
END
```

```
-----
ERR_HANDLER:
    --SQL Error log ID rtned to the calling app. Or -1 failure to log.
    execute @iErrLogID = usp_DBA_LogError @piAppID, @iUspID, @iErrnum, @iStep,
    @iRowCount
    return @iErrLogID
```

14.2 usp_DBA_LogError

This proc has already been created in each DB & shouldn't be altered.

CREATE proc **usp_DBA_LogError** @piAppID int,@piUspID int, @piErrNum int,@piStep int,
@piRowCnt int as

/******

Procedure

usp_DBA_LogError

Purpose

Logs usp errors to local SQLErrorLog tbl

Revision History

1.0 29/4/02 KB Created

Input Params

@piAppID = application ID (see platinfo tbl)

@piUspID = stored proc id

@piErrNum = error num

@piStep = crash point

@piRowCnt = rowcount

Output Params

none

SQL Return Code

0 success (this usp will rtn a positive integer is successful not 0)

1 up SQLErrorlog tbl unique row ID

-1 This usp failed

Dependencies

Tables: BusinessLogicMessages, SQLErrorLog

Procs:

Called by

Apps:

USPs:

Jobs:

*****/

SET NOCOUNT ON

declare @dtErrTime datetime

declare @vcDBuser varchar(128)

declare @vcLogin varchar(128)

declare @vcUspName varchar(150)

declare @vcExeName varchar(150)

declare @iSPID int

declare @vcHostName varchar(50)

declare @vcErrMsg varchar(400)

declare @iErrLogID int

declare @vcSvrName varchar(30)

declare @vcDBName varchar(50)

declare @iDebug int

set @iDebug = 1

--Print input params

if @iDebug = 1

begin

print @piAppID print @piUspID print @piErrNum print @piStep print @piRowCnt

end

```
-----
--Check input values & remove NULLS
if @piAppID is null set @piAppID = 0    --Not in app tbl
if @piUspID is null set @piUspID = 0    --Set uspname later
if @piErrNum is null set @piErrNum = 999999    --Dummy value to signify NULL
if @piStep is null set @piStep = 0
if @piRowCnt is null set @piRowCnt = 0

set @dtErrTime = getdate()

set @vcDBUser = (select user)
if @vcDBUser = null set @vcDBUser = 'Unknown'

set @vcLogin = (select SYSTEM_USER)
if @vcLogin is null set @vcLogin = 'Unknown'

set @vcSvrName = left(@@SERVERNAME,30)    --Which SQL svr
if @vcSvrName is null set @vcSvrName = 'Unknown'

set @vcDBName = left(db_name(),50)    --Which DB
if @vcDBName is null set @vcDBName = 'Unknown'

--Current sys process ID
set @iSPID =(SELECT @@SPID)
if @iSPID is null set @iSPID = 0

--Find exe / program name
set @vcExeName = (select program_name from master.dbo.sysprocesses where SPID=@iSPID)
if @vcExeName is null set @vcExeName = 'Unknown'

--PC executed from
set @vcHostName = (select hostname from master.dbo.sysprocesses where SPID=@iSPID)
if @vcHostName is null set @vcHostName = 'Unknown'

--Find sp name
IF @piUspID = 0
    set @vcUspName = 'UNKNOWN'
ELSE
    set @vcUspName = (select name from sysobjects where id = @piUspID)

-----
IF @@error > 0 GOTO ERR_HANDLER -- ie setting the var values failed somewhere, so don't
log.
BEGIN
    --Find err msg
    select @vcErrMsg=
        case
            when @piErrNum = 999999 then 'NULL error ID'
            when @piErrNum = 0 then 'Success'
            when @piErrNum >0 and @piErrNum < 50001 then (select description
from master.dbo.sysmessages where error = @piErrNum)
            when @piErrNum >50001 then (select descr from
BusinessLogicMessages where BLM_ID = @piErrNum)
            else 'none'
```

```

                end
END
-----
if @iDebug =1
    BEGIN
        print @dtErrTime print @vcDBUser print @vcLogin print @vcSvrName print
@vcDBName
        print @iSPID print @vcExeName print @vcHostName print @vcUspName print
@vcErrMsg
    END
-----
IF @@error > 0 GOTO ERR_HANDLER -- ie setting the errmsg or debug failed somewhere, so
don't log.
BEGIN
    insert into SQLErrorLog
    (
        AppID,
        ExeName,
        HostName,
        dbuser,
        Login,
        usp,
        ErrTime,
        ErrNum,
        ErrMsg,
        Step,
        Rowcnt,
        SvrName,
        DBname
    )
    values
    (
        @piAppID,
        @vcExeName,
        @vcHostName,
        @vcDBUser,
        @vcLogin,
        @vcUspName,
        @dtErrTime,
        @piErrNum,
        @vcErrMsg,
        @piStep,
        @piRowCnt,
        @vcSvrName,
        @vcDBname
    )
END
-----
IF @@error > 0 GOTO ERR_HANDLER --Insert into SQL error log failed, thus no errlogID to
rtn.
BEGIN
    --Rtn err log ID to calling proc.
    --No NULLS or breaks comparisons.
    select @iErrLogID = ErrorId from SQLErrorLog
    where
```

```
AppID=@piAppID and  
ExeName=@vcExeName and  
HostName=@vcHostName and  
dbuser=@vcDBuser and  
usp=@vcUspName and  
ErrTime=@dtErrTime and  
ErrNum=@piErrNum and  
Step=@piStep and  
Rowcnt=@piRowCnt
```

END

IF @@error > 0 OR @@rowcount <> 1 GOTO ERR_HANDLER --Could not find SQL error log ID.

BEGIN

Return @iErrLogID

END

ERR_HANDLER:

--Failed
return -1

GO

14.3 Objects created locally in each DB

These objects must be created locally in each DB. Most are already created, & the DBA can create these when a new DB is created. Or put in Model db.

```
CREATE TABLE [dbo].[SQLErrorLog] (  
    [ErrorId] [int] IDENTITY (1, 1) NOT NULL ,  
    [AppID] [int] NOT NULL ,  
    [ExeName] [varchar] (150) NOT NULL ,  
    [HostName] [varchar] (50) NOT NULL ,  
    [dbuser] [varchar] (128) NOT NULL ,  
    [SvrName] [varchar] (30) NOT NULL ,  
    [DBname] [varchar] (50) NOT NULL ,  
    [usp] [varchar] (150) NOT NULL ,  
    [ErrTime] [datetime] NOT NULL ,  
    [ErrNum] [int] NOT NULL ,  
    [ErrMsg] [varchar] (255) NOT NULL ,  
    [Step] [int] NOT NULL ,  
    [Rowcnt] [int] NOT NULL ,  
    [Login] [varchar] (128) NOT NULL  
) ON [PRIMARY]  
GO
```

```
create rule [rule_BLM_type] as @colvalue = 'success' or @colvalue = 'failure'  
GO
```

```
CREATE TABLE [dbo].[BusinessLogicMessages] (  
    [BLM_ID] [int] IDENTITY (50001, 1) NOT NULL ,  
    [Descr] [varchar] (255) NULL ,  
    [BLM_Type] [char] (10) NULL  
) ON [PRIMARY]  
GO
```

```
setuser N'dbo'  
GO
```

```
EXEC sp_bindrule N'[dbo].[rule_BLM_type]', N'[BusinessLogicMessages].[BLM_Type]'  
GO
```

15 Generating test errors

These were used to test the error handling process.

15.1 FK error

```
select * from titleauthor  
where title_id = 'PS3333'
```

```
insert into titleauthor (au_id,title_id) values('172-32-1176','PS3333')  
select @@error, @@rowcount
```

```
Server: Msg 2627, Level 14, State 1, Line 0  
Violation of PRIMARY KEY constraint 'UPKCL_taind'. Cannot insert duplicate key in object  
'titleauthor'.  
The statement has been terminated.
```

```
-----  
2627      0  
(1 row(s) affected)
```

15.2 Identity insert error

```
Select * from customer  
where custid = '2330'
```

```
insert into customer (custid) values('2330')  
select @@error, @@rowcount
```

```
Server: Msg 544, Level 16, State 1, Line 0  
Cannot insert explicit value for identity column in table 'Customer' when IDENTITY_INSERT is set  
to OFF.
```

```
-----  
544      0  
(1 row(s) affected)
```

16 Code that receives the error info.

The following code egs, demonstrate how to use the return code & multiple output parameters to receive error info, as well as the main resultset.

16.1 Javascript - ASP

http://.intranet.com/data/test/adodemo_js.asp

```
<h2>
<font face="Arial, Helvetica, sans-serif">ASP Demo to show error trapping methods</font>
</h2>
<p>
<%@ Language=JavaScript %>
<%
var cmd = Server.CreateObject ("ADODB.Command")
var rs = Server.CreateObject ("ADODB.Recordset")
cmd.CommandText = "usp_ADOTest"
cmd.CommandType = 4
cmd.ActiveConnection = "Driver={SQL Server};server=terra;database=pubs;UID=reporters;pwd=bla;"
cmd.Parameters.Refresh
cmd.Parameters(1).Value = "%e%"
rs = cmd.Execute ()
while (!rs.EOF)
{
    Response.Write (rs.Fields(0).Value + "")
    Response.Write (rs.Fields(1).Value + "")
    Response.Write ("<BR>")
    rs.MoveNext()
}
rs.Close()
Response.Write ("<BR>")
Response.Write ("Ret param=" + cmd.Parameters(0).Value + "")
Response.Write ("<BR>")
Response.Write ("in param=" + cmd.Parameters(1).Value + "")
Response.Write ("<BR>")
Response.Write ("out param=" + cmd.Parameters(2).Value)
Response.Write ("<BR>")
Response.Write ("out param 2=" + cmd.Parameters(3).Value)
%>
</p>
```

16.2 Vbscript – ASP

<http://.intranet.com/data/test/adodemo.asp>

```
<h2>
<font face="Arial, Helvetica, sans-serif">ASP Demo to show error trapping methods</font>
</h2>
<p>
<%
dim cmd
set cmd = Server.CreateObject ("ADODB.Command")
set rs = Server.CreateObject ("ADODB.Recordset")
cmd.CommandText = "usp_ADOTest"
cmd.CommandType = &H0004
cmd.ActiveConnection = "Driver={SQL Server};server=terra;database=pubs;UID=reporters;pwd=bla;"
cmd.Parameters.Refresh
cmd.Parameters(1).Value = "%e%"
Set rs = cmd.Execute ()
rs.movefirst
do while not rs.eof
    Response.Write (rs(0)) & " "
    Response.Write (rs(1)) & "" & "<BR>"
    rs.MoveNext
loop
rs.Close()
Response.Write ("SQL return param=" & cmd.Parameters(0).Value & "" & "<BR>" )
Response.Write ("input param=" & cmd.Parameters(1).Value & "" & "<BR>" )
Response.Write ("output param=" & cmd.Parameters(2).Value & "<BR>" )
Response.Write ("output param=" & cmd.Parameters(2).Value & "<BR>" )
%>
</p>
```

16.3 TSQL

This piece of TSQL, provides the same functionality as the other script code egs.

To execute a stored procedure from another sp.
Pass in the app ID as an input parameter,
provided by the application or from platforminfo..applications tbl.
Receive return code & output parameter.

```
Declare @iRtnCode int
Declare @iErrnum int
Declare @iErrnum2 int

execute @iRtnCode = usp_TestProc @piAppID, @piErrnum = @iErrnum OUTPUT, @piErrnum2 = @iErrnum2
OUTPUT

print @iErrnum print @iErrnum2 print @iRtnCode
```

To test the suggested logging method, use the following:

```
Declare @iRtnCode int

execute @iRtnCode = usp_DBA_Template_LogErrors 2
print @iRtnCode

select * from sqlerrorlog
```

16.4 C++

When calling a SQL sp, C++ receives OLEDB error numbers, which can provide a looked up message.
These can be logged to a file.

17 Retrieving the error info

In your C++ or ASP app, call this usp to retrieve the full SQL error info.

@piErrLogID = PK in SQLerrorlog tbl
@piAppID see platinfo.applications.

You received the @piErrLogID in the return code when you called your usp.

```
usp_DBA_GetSQLerrInfo @piAppID int, @piErrLogID
```

The resultset will be 1 row

18 References

BOL lookup 'dbretstatus'
SQL Pro Mag author: Sunderic
MSDN