# SQL Server Structure Error Handling

Add rich reporting of exceptions to improve error reporting to users and diagnostics to development.

## Introduction

SQL features for report exceptions are very limiting; all that is really available is the single text string passed as an argument to the “RAISERROR” error statement. As a developer I would very much like to include the information below, but require very little coding when reporting the error.

* A unique error number to identify the error. This is required for application documentation, where a list of error message and some unique error number is commonly required.
* A message that can be displayed to a general user of the application.
* A specific message to a developer using the SQL code what the problem is.
* Any relevant variable data or status information.
* Name of the procedure reporting the error.
* Trace back information of the procedure calls to the point of exception.

## Approach

SQL really only provides two basic tools for reporting and processing exceptions, “RAISERROR” to report exceptions and TRY-CATCH block for exception processing. “RAISERROR” is limited to a single 2047 character string, an ambiguous severity code between 0 and 18 and an even more ambiguous state code between 0 and 255. When constructing the single line of text for “RAISERROR”, one has to be careful what is included in the message because many times it is displayed directly to the end user. The “TRY-CATCH” block first introduced in SQL Server 2008, provides a very powerful tool to capture and consolidate exception handling, report the procedure and line where the exception occurred and manage trace back of procedure calls.

I propose the best approach is to construct an XML document containing the error information required and send the XML document as a string within the “RAISERROR” mechanism. In my design, a typical basic error message looks like the following.

<E N="2021003" M="The Branch specified was not found"  
 D="Specified branch id not found or deleted."  
 P="ArticleIsDeletable" L="103">  
 <T EntityId="15" EntityType="2"/>  
</E>

This has all the rich information I need to send to the caller all in a structured format that can be easily captured by the caller, which can then extract the information as required. We find it is extremely useful when these XML documents appear in any diagnostic or log output. This XML document contains the following:

* The unique error number as attribute “N”, i.e. “2021003”. This is a common need in the industry for unique error numbers with error messages to provide to customers.
* A message to display to the user as attribute “M”.
* A message to display to the developer as attribute “D”.
* Name of the procedure and line number as attribute “P” and “L”.
* Special token variable and status information in element “T”, where there is some data name “EntityId” and “EntityType” with the values of 15 and 2 respectively.

With this construct, a caller can encapsulate this error within its own error message such as this.

<E N="1041000"  
 M="The branch was most likely deleted by someone else, please refresh display."  
 D="Cannot find source branch."   
 P="DeleteArticle" L="123">  
 <T ArticleId="15" ArticleType="2"/>  
 <E N="2021003" M="The Branch specified was not found"  
 D="Specified branch id not found or deleted."   
 P="ArticleIsDeletable" L="103">  
 <T EntityId="15" EntityType="2"/>  
 </E>  
</E>

Which immediately lets you know the store procedure “DeleteArticle” was called with article identifier 15 and type 2 and that it failed at line 123 when calling store procedure “ArticleIsDeletable”, which failed at line 103. The user interface can easily extract a useful error message to the user, but still log detail information on the error. Our applications have even included a “Diagnostic Message” button in error panels that provide the full diagnostic information in the error structure.

The element and attribute names were chosen to minimize the number of characters used, since the resulting XML has to be serialized into a string which has to fit within the 2047 character limitation of the “RAISERROR” statement. I have found that a lot of information can be stored within this 2047 character limitation by utilizing this structure mechanism as long as you can keep the XML text bloat to a minimum. This structured format liberated us to provide a reasonable end user message while still provide all sorts of diagnostic information, which has proven to be very effective in tracking down problems quickly.

## Design

The magic to implement structured error messages is now made possible with the extensive support for XML that was first introduced in SQL Server 2005. I wrote a series of procedures that simplify implementation of error handling within SQL code, for example the following sample code shows the basics on how to throw, capture and process errors.

CREATE PROCEDURE dbo.GetMeterEntityStatus

@intParam int

, @stringParam varchar(max)

, @dateTimeParam datetime

AS

BEGIN TRY

DECLARE @RC int = 0

, @ErrorMessage varchar(max)

;

-- Define procedure code here. --

IF @stringParam IS NULL

BEGIN

-- Some error occurred.

SET @ErrorMessage = StructuredError.ErrorLookup(@@PROCID, 'SomeError',

(SELECT \* FROM (SELECT intParam=@intParam,  
 stringParam=@stringParam,  
 dateTimeParam=@dateTimeParam  
 ) AS T FOR XML AUTO  
 ));

RAISERROR(@ErrorMessage, 16, 10);

END

-- Define procedure code here. --

RETURN @RC;

END TRY

BEGIN CATCH

EXEC StructuredError.ErrorHandler @@PROCID

END CATCH

The “TRY-CATCH” block is a powerful tool to capture all exceptions thrown and provide an appropriate structured error message. I have found employing “TRY-CATCH” in all store procedures provides excellent control and management of exception handling. The store procedure “ErrorHandler” provides all the logic necessary to package and throw the error messages as required. Utilizing the system functions “ERROR\_MESSAGE”, “ERROR\_PROCEDURE” and “ERROR\_LINE”, this store procedure determines if the message is already a structured error message or an ordinary message and takes appropriate action to enhance the information and structure it to be thrown with appropriate information. In this way existing system error message and a simple thrown message are handled appropriately. The store procedure “ErrorHandler” also has a feature to allow for transaction cleanup operations in case it is necessary to invoke certain SQL commands. To do this, just capture the error information in an XML string, perform cleanup and then pass the XML string to “ErrorHandler” like in the example below.

DECLARE @ThrownErrorCapture VARCHAR(max)

= (SELECT \* FROM (SELECT ErrorNumber=ERROR\_NUMBER(),

ErrorMessage=ERROR\_MESSAGE(),

ErrorProcedure=ERROR\_PROCEDURE(),

ErrorLine=ERROR\_LINE(),

ErrorSeverity=ERROR\_SEVERITY(),

ErrorState=ERROR\_STATE()

) AS T FOR XML AUTO) ;

IF @@trancount > 0 ROLLBACK;

-- Some cleanup operations here. --

EXEC StructuredError.ErrorHandler @@PROCID, NULL, @ThrownErrorCapture

When constructing a custom structured error message to be thrown with “RAISERROR”, the user defined function “ErrorLookup” provides the logic necessary to lookup an error message text from an error table, construct a structured error message and include optional token data. This function also ensures the resulting text fits within the 2047 character limitation of the “RAISERROR” statement.

## Auditing

Procedure call auditing is optionally supported, which records name of procedure, start time, input data, output data, error messages and end time. These are facilitated by the procedures “BeginAuditLogProcEntry”, “EndAuditLogEntry” and an optional parameter to “ErrorHandler”. An expanded definition of the example store procedure “GetMeterEntityStatus” with auditing is shown below.

CREATE PROCEDURE dbo.GetMeterEntityStatus

@intParam int

, @stringParam varchar(max)

, @dateTimeParam datetime

AS

BEGIN TRY

DECLARE @True bit = 1

, @False bit = 0

;

DECLARE @RC int = 0

, @AuditLogId bigint = NULL

, @ErrorMessage varchar(max)

, @xmlParams xml

= (SELECT \* FROM (SELECT intParam=@intParam,

stringParam=@stringParam,

dateTimeParam=@dateTimeParam

) AS params FOR XML AUTO)

;

EXEC StructuredError.BeginAuditLogProcEntry @AuditLogId OUTPUT, @@PROCID, @True, @xmlParams

;

-- Define procedure code here. --

IF @stringParam IS NULL

BEGIN

-- Some error occurred.

SET @ErrorMessage = StructuredError.ErrorLookup(@@PROCID, 'SomeError',

(SELECT \* FROM (SELECT intParam=@intParam,

stringParam=@stringParam,

dateTimeParam=@dateTimeParam

) AS T FOR XML AUTO

));

RAISERROR(@ErrorMessage, 16, 10);

END

-- Define procedure code here. --

EXEC StructuredError.EndAuditLogEntry @AuditLogId

;

RETURN @RC;

END TRY

BEGIN CATCH

EXEC StructuredError.ErrorHandler @@PROCID, @AuditLogId

END CATCH

Whenever an exception occurs, having the original arguments and full structured error message in the auditing is very beneficial for diagnostic purposes. Many times the rich error information alone is able to isolate the issue.

## License

For legal considerations, these source files are free software, under a BSD style license. These source file is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the name of Erickson and Associates may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY Erickson and Associates "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL Erickson and Associates BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.